Fortran and C interoperability with modern Fortran

Patrick Bégou
Patrick.Begou@univ-grenoble-alpes.fr

GT-Fortran / October 28^{th} 2026

Motivation

Why?

- Many C libraries are available for use in Fortran code
- Using Fortran kernels in C applications

A basic example

- A set of sample values $x_i \ldots$
- We need to calculate the estimated variance:

$$\sigma^2 = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \overline{x})^2$$

- **Good:** The **G**NU **S**cientific **L**ibrary (**GSL**) provides the *gsl_stats_variance(...)* function for floating-point data in the *libgsl* library.
- **Bad:** *libgsl* is a C library, Fortran interfacing is still under development.

Basic C approach

gsl_stats_variance

```
double gsl_stats_variance(const double data[],
                          size_t stride, size_t n)
```

Simple C implementation in main.c

```
1  #include <gsl/gsl_statistics_double.h>
2  #include <stdio.h>
3  int main() {
4    double tab[3]={9.0, 10.0, 11.0};
5    double variance=0.0;
6
7    variance = gsl_stats_variance(tab, 1, 3);
8
9    printf("Variance: %14.6f \n",variance);
10   return 0;
11 }
```

Building and running the C application

Using GNU compilers

```
bash-4.4$ gcc -c main.c
bash-4.4$ gcc -o mainc.exe main.o  -lgsl -lgslcblas -lm
bash-4.4$ ./mainc.exe
Variance:         1.000000
bash-4.4$
```

Building and running the C application

Using GNU compilers

```
bash-4.4$ gcc -c main.c
bash-4.4$ gcc -o mainc.exe main.o  -lgsl -lgslcblas -lm
bash-4.4$ ./mainc.exe
Variance:         1.000000
bash-4.4$
```

Let's go with Fortran application now!



Legacy Fortran approach

A deliberately naive Fortran implementation

```
1  program main
2     implicit none
3     integer, parameter :: n=3
4     double precision, dimension(n) :: tab=(/9.0, 10.0, 11.0/)
5     double precision :: variance
6
7     double precision gsl_stats_variance
8
9     variance = gsl_stats_variance(tab, 1, n)
10
11    write(6,'(a,1x,f12.4)')"Variance:",variance
12
13 end program main
```

- *line 7* type of `gsl_stats_variance` is defined
- No explicit interface for `gsl_stats_variance`

Legacy Fortran approach

A deliberately naive Fortran implementation

```
1  program main
2     implicit none
3     integer, parameter :: n=3
4     double precision, dimension(n) :: tab=(/9.0, 10.0, 11.0/)
5     double precision :: variance
6
7     double precision gsl_stats_variance
8
9     variance = gsl_stats_variance(tab, 1, n)
10
11    write(6,'(a,1x,f12.4)')"Variance:",variance
12
13 end program main
```

Compilers agree

bash-4.4: gfortran -c main.f90
bash-4.4: ifx -c main.f90
bash-4.4: nvfortran -c main.f90

Legacy Fortran approach

A deliberately naive Fortran implementation

```
1  program main
2     implicit none
3     integer, parameter :: n=3
4     double precision, dimension(n) :: tab=(/9.0, 10.0, 11.0/)
5     double precision :: variance
6
7     double precision gsl_stats_variance
8
9     variance = gsl_stats_variance(tab, 1, n)
10
11    write(6,'(a,1x,f12.4)')"Variance:",variance
12
13 end program main
```

Compilers agree

bash-4.4: gfortran -c main.f90
bash-4.4: ifx -c main.f90
bash-4.4: nvfortran -c main.f90

Linkers do not agree (trailing "_")

```
bash-4.4: gfortran main.o \
                        -lgsl -lgslcblas -lm
main.o: In function ‘MAIN__’:
main.f90:(.text+0x1b): undefined
reference to gsl_stats_variance_
```

Legacy Fortran approach

Using a [bad] C wrapper: *wrapper.c*

```c
#include <gsl/gsl_statistics_double.h>
double gsl_stats_variance_ (const double data[],
                            size_t stride,
                            size_t n) {
    return gsl_stats_variance(data, stride, n);
}
```

- same signature as the original API of `gsl_stats_variance`
- "_" added to the function name
- used as a wrapper function

Legacy Fortran approach

Using a [bad] C wrapper: *wrapper.c*

```c
#include <gsl/gsl_statistics_double.h>
double gsl_stats_variance_ (const double data[],
                            size_t stride,
                            size_t n) {
    return gsl_stats_variance(data, stride, n);
}
```

Compilers agree

bash-4.4: gfortran -c main.f90
bash-4.4: gcc -c wrapper.c
gbash-4.4: fortran main.o wrapper.o -o mainf.exe -lgsl -lgslcblas -lm

---

bash-4.4: ifx -c main.f90
bash-4.4: icx -c wrapper.c
bash-4.4: ifx main.o wrapper.o -o mainf.exe -lgsl -lgslcblas -lm

---

bash-4.4: nvfortran -c main.f90
bash-4.4: nvc -c wrapper.c
bash-4.4: nvfortran main.o wrapper.o -o mainf.exe -lgsl -lgslcblas -lm

Legacy Fortran approach

Using a [bad] C wrapper: *wrapper.c*

```c
#include <gsl/gsl_statistics_double.h>
double gsl_stats_variance_ (const double data[],
                            size_t stride,
                            size_t n) {
    return gsl_stats_variance(data, stride, n);
}
```

Execution do not agree

bash-4.4: ./mainf.exe

Program received signal SIGSEGV: Segmentation fault - invalid memory reference.

Execution do not agree because ...

Parameters `data`, `stride` and `n` are not correctly handled between C and Fortran

Legacy Fortran approach

Argument types and interoperability

- A double-precision array in Fortran is interoperable with an array of doubles in C
- A Fortran integer i
  - must satisfy $-2^{31} + 1 \leq i \leq +2^{31} - 1$
  - is typically 32 bits wide
  - is signed
- A C `size_t` variable i
  - can represent the size of any object
  - typically supports $0 \leq i \leq 2^{64} - 1$ on 64-bit systems
  - is typically 64 bits wide on 64-bit systems
  - is unsigned

Arguments passing

- For the array, C and Fortran send/receive the address of the first element.
- Fortran scalar arguments
  - sent/received by address (pointers)
  - In *main.f90* the addresses of 1 and `n` are sent to `gsl_stats_variance_(...)`
- C scalar arguments
  - sent/received by value
  - In *wrapper.c* `gsl_stats_variance_(...)` receives addresses for `stride` and `n` instead of their values!

Legacy Fortran approach

Using a working C wrapper: *wrapper.c*

```c
#include <gsl/gsl_statistics_double.h>
double gsl_stats_variance_(const double data[],
                           int* stride, int* n) {
    if (*stride < 0 || *n < 0) {
        /* manage the error */
        return 0.0;
    }
    return gsl_stats_variance(data, (size_t) *stride, (size_t) *n);
}
```

- scalar arguments received as pointers to int
- here, C `int` is interoperable with Fortran `integer`
- optional `size_t` cast for the values of `stride` and `n` in the final call

Legacy Fortran approach

Using a working C wrapper: *wrapper.c*

```c
#include <gsl/gsl_statistics_double.h>
double gsl_stats_variance_(const double data[],
                           int* stride, int* n) {
    if (*stride < 0 || *n < 0) {
        /* manage the error */
        return 0.0;
    }
    return gsl_stats_variance(data, (size_t) *stride, (size_t) *n);
}
```

- scalar arguments received as pointers to int
- here, C `int` is interoperable with Fortran `integer`
- optional `size_t` cast for the values of `stride` and `n` in the final call

Compilation works

bash-4.4: nvfortran -c main.f90
bash-4.4: nvc -c wrapper.c
bash-4.4: nvfortran main.o wrapper.o -o mainf.exe -lgsl -lgslcblas -lm

Legacy Fortran approach

Using a working C wrapper: *wrapper.c*

```c
#include <gsl/gsl_statistics_double.h>
double gsl_stats_variance_(const double data[],
                           int* stride, int* n) {
    if (*stride < 0 || *n < 0) {
        /* manage the error */
        return 0.0;
    }
    return gsl_stats_variance(data, (size_t) *stride, (size_t) *n);
}
```

- scalar arguments received as pointers to int
- here, C `int` is interoperable with Fortran `integer`
- optional `size_t` cast for the values of `stride` and `n` in the final call

Execution works

bash-4.4: ./mainf.exe
Variance: 1.0000

Legacy Fortran approach

Using a working C wrapper: *wrapper.c*

```c
#include <gsl/gsl_statistics_double.h>
double gsl_stats_variance_(const double data[],
                           int* stride, int* n) {
    if (*stride < 0 || *n < 0) {
        /* manage the error */
        return 0.0;
    }
    return gsl_stats_variance(data, (size_t) *stride, (size_t) *n);
}
```

Result

- Compilers agree, linkers agree, and execution succeeds
- But it is complicated and still provides no argument checking
- Let's investigate modern **Fortran ISO_C_BINDING** approach!

Execution works

bash-4.4: ./mainf.exe
Variance: 1.0000

Modern Fortran approach

Fortran–C interoperability

The **Fortran ISO_C_BINDING** approach provides:

- Standardized mechanism for operating with C
- Interoperable intrinsic types
- Interoperability of arrays
- Interoperability of functions/subroutines
- Interoperability of derived types (limited)
- Management of C and Fortran pointers
- . . . etc.

"Code"

```
use, intrinsic :: iso_c_binding
```

Fortran ISO_C_BINDING

- C wrappers are no longer required for these cases
- Function argument checking at compile time
- Calling C from Fortran
- Calling Fortran from C

Modern Fortran approach

Some interoperable intrinsic types

Type	Named constant	C type
integer	c_int	int
	c_short	short int
	c_long	long int
	c_long_long	long long int
	c_size_t	size_t
real	c_float	float
	c_double	double
	c_long_double	long double
logical	c_bool	_Bool
character	c_char	char

"C/Fortran interoperable variables"

```
integer(c_int) :: my_fc_int     !a 4 bytes integer variable
real(c_double) :: my_fc_real    !a 8 bytes floating point variable
```

Modern Fortran approach

Fortran calling C

```
program main
  use, intrinsic :: iso_c_binding
  implicit none

  interface
    subroutine swap(a,b) bind(c)
       use, intrinsic :: iso_c_binding
       implicit none
       integer(c_int), intent(inout) :: a,b
    end subroutine swap
  end interface

  integer(c_int) :: tab(5) = (/1,2,3,4,5/)

  write(6,'("[ ",5(i0,1x),"]")') tab
  call swap(tab(1), tab(5))
  write(6,'("[ ",5(i0,1x),"]")') tab

end program main
```

Minimal C function

```
void swap(int* a, int* b)
{
   int c;
   c=*a,  *a=*b, *b=c;
}
```

Modern Fortran approach

Fortran calling C

```
program main
  use, intrinsic :: iso_c_binding
  implicit none

  interface
    subroutine swap(a,b) bind(c)
       use, intrinsic :: iso_c_binding
       implicit none
       integer(c_int), intent(inout) :: a,b
    end subroutine swap
  end interface

  integer(c_int) :: tab(5) = (/1,2,3,4,5/)

  write(6,'("[ ",5(i0,1x),"]")') tab
  call swap(tab(1), tab(5))
  write(6,'("[ ",5(i0,1x),"]")') tab

end program main
```

Minimal C function

```
void swap(int* a, int* b)
{
   int c;
   c=*a,  *a=*b, *b=c;
}
```

Inside ISO_C_BINDING

- Load module iso_c_binding
- Explicit interface for swap()
- `bind(c)` attribute
- Interoperable arguments
- Arguments are passed according to the Fortran standard (by address)

Modern Fortran approach

Fortran calling C

```
program main
  use, intrinsic :: iso_c_binding
  implicit none

  interface
    subroutine swap(a,b) bind(c)
       use, intrinsic :: iso_c_binding
       implicit none
       integer(c_int), intent(inout) :: a,b
    end subroutine swap
  end interface

  integer(c_int) :: tab(5) = (/1,2,3,4,5/)

  write(6,'("[ ",5(i0,1x),"]")') tab
  call swap(tab(1), tab(5))
  write(6,'("[ ",5(i0,1x),"]")') tab

end program main
```

Minimal C function

```
void swap(int* a, int* b)
{
   int c;
   c=*a,  *a=*b, *b=c;
}
```

Inside ISO_C_BINDING

- Load module iso_c_binding
- Explicit interface for swap()
- `bind(c)` attribute
- Interoperable arguments
- Arguments are passed according to the Fortran standard (by address)

Modern Fortran approach

Fortran calling C

```
program main
  use, intrinsic :: iso_c_binding
  implicit none

  interface
    subroutine swap(a,b) bind(c)
       use, intrinsic :: iso_c_binding
       implicit none
       integer(c_int), intent(inout) :: a,b
    end subroutine swap
  end interface

  integer(c_int) :: tab(5) = (/1,2,3,4,5/)

  write(6,'("[ ",5(i0,1x),"]")') tab
  call swap(tab(1), tab(5))
  write(6,'("[ ",5(i0,1x),"]")') tab

end program main
```

Minimal C function

```
void swap(int* a, int* b)
{
   int c;
   c=*a,  *a=*b, *b=c;
}
```

Inside ISO_C_BINDING

- Load module iso_c_binding
- Explicit interface for swap()
- `bind(c)` attribute
- Interoperable arguments
- Arguments are passed according to the Fortran standard (by address)

Modern Fortran approach

Fortran calling C

```
program main
  use, intrinsic :: iso_c_binding
  implicit none

  interface
    subroutine swap(a,b) bind(c)
       use, intrinsic :: iso_c_binding
       implicit none
       integer(c_int), intent(inout) :: a,b
    end subroutine swap
  end interface

  integer(c_int) :: tab(5) = (/1,2,3,4,5/)

  write(6,'("[ ",5(i0,1x),"]")') tab
  call swap(tab(1), tab(5))
  write(6,'("[ ",5(i0,1x),"]")') tab

end program main
```

Minimal C function

```
void swap(int* a, int* b)
{
   int c;
   c=*a,  *a=*b, *b=c;
}
```

Inside ISO_C_BINDING

- Load module iso_c_binding
- Explicit interface for swap()
- `bind(c)` attribute
- Interoperable arguments
- Arguments are passed according to the Fortran standard (by address)

Modern Fortran approach

Fortran calling C

```
program main
  use, intrinsic :: iso_c_binding
  implicit none

  interface
    subroutine swap(a,b) bind(c)
       use, intrinsic :: iso_c_binding
       implicit none
       integer(c_int), intent(inout) :: a,b
    end subroutine swap
  end interface

  integer(c_int) :: tab(5) = (/1,2,3,4,5/)

  write(6,'("[ ",5(i0,1x),"]")') tab
  call swap(tab(1), tab(5))
  write(6,'("[ ",5(i0,1x),"]")') tab

end program main
```

Minimal C function

```
void swap(int* a, int* b)
{
    int c;
    c=*a,  *a=*b, *b=c;
}
```

Inside ISO_C_BINDING

- Load module iso_c_binding
- Explicit interface for swap()
- `bind(c)` attribute
- Interoperable arguments
- Arguments are passed according to the Fortran standard (by address)

Modern Fortran approach

GNU Fortran 16.1.0

```
gfortran -c main.f90
gcc -c swap.c
gfortran main.o swap.o -o main.exe

./main.exe
[ 1 2 3 4 5 ]
[ 5 2 3 4 1 ]
```

NVIDIA Fortran 26.5-0

```
nvfortran -c main.f90
nvc -c swap.c
nvfortran main.o swap.o -o main.exe

./main.exe
[ 1 2 3 4 5 ]
[ 5 2 3 4 1 ]
```

Intel Fortran 2026.1

```
ifx -c main.f90
icx -c swap.c
ifx main.o swap.o -o main.exe

./main.exe
[ 1 2 3 4 5 ]
[ 5 2 3 4 1 ]
```

Cray Fortran 20.0

```
ftn -c main.f90
cc -c swap.c
ftn main.o swap.o -o main.exe

./main.exe
[ 1 2 3 4 5 ]
[ 5 2 3 4 1 ]
```

Modern Fortran approach

Fortran calling C

```
program main
  use, intrinsic :: iso_c_binding
  implicit none

  interface
    subroutine swap(a,b) bind(c)
       use, intrinsic :: iso_c_binding
       implicit none
       integer(c_int), intent(inout) :: a,b
    end subroutine swap
  end interface

  integer :: tab(5) = (/1,2,3,4,5/)

  write(6,'("[ ",5(i0,1x),"]")') tab
  call swap(tab(1), tab(5))
  write(6,'("[ ",5(i0,1x),"]")') tab

end program main
```

Minimal C function

```
void swap(int* a, int* b)
{
   int c;
   c=*a,  *a=*b, *b=c;
}
```

Integer intrinsic type

`integer` intrinsic type is interoperable with `integer(c_int)` type.

```
nvfortran -c main.f90
nvc -c swap.c
nvfortran main.o swap.o \
          -o main.exe

./main.exe
[ 1 2 3 4 5 ]
[ 5 2 3 4 1 ]
```

Modern Fortran approach

Fortran calling C

```
1  program main
2    use, intrinsic :: iso_c_binding
3    implicit none
4
5    interface
6      subroutine swap(a,b) bind(c)
7         use, intrinsic :: iso_c_binding
8         implicit none
9         integer(c_int), intent(inout) :: a,b
10     end subroutine swap
11   end interface
12
13   real :: tab(5) = (/1.0,2.0,3.0,4.0,5.0/)
14
15   write(6,'("[ ",5(i0,1x),"]")') tab
16   call swap(tab(1), tab(5))
17   write(6,'("[ ",5(i0,1x),"]")') tab
18
19
```

Minimal C function

```
void swap(int* a, int* b)
{
   int c;
   c=*a,  *a=*b, *b=c;
}
```

Arguments are checked

```
nvfortran -c main.f90
NVFORTRAN-S-0188-Argument number 1 to swap: type mismatch (main.f90: 16)
NVFORTRAN-S-0188-Argument number 2 to swap: type mismatch (main.f90: 16)
```

gsl_stats_variance() with ISO_C_BINDING

gsl_stats_variance

```
double gsl_stats_variance(const double data[],
                          size_t stride, size_t n)
```

Fortran calling C

```
program main
   use, intrinsic:: iso_c_binding
   implicit none
   integer(c_size_t), parameter :: n=3
   real(c_double), dimension(n) :: tab=(/9.0, 10.0, 11.0/)
   real(c_double) :: variance

   interface
     function gsl_stats_variance(data,stride,n) &
                 & bind(c, name="gsl_stats_variance")
       use, intrinsic:: iso_c_binding
       implicit none
       real(c_double) :: gsl_stats_variance
       real(c_double), intent(in) :: data(*)
       integer(c_size_t), value :: stride, n
     end function gsl_stats_variance
   end interface

   variance = gsl_stats_variance(tab, 1_c_size_t, n)

   write(6,'(a,1x,f12.4)')"Variance:",variance

end program main
```

Back to initial problem

- C wrapper removed
- `iso_c_binding` module loaded
- Interface bloc provided

gsl_stats_variance() with ISO_C_BINDING

gsl_stats_variance

```
double gsl_stats_variance(const double data[],
                          size_t stride, size_t n)
```

Fortran calling C

```
program main
   use, intrinsic:: iso_c_binding
   implicit none
   integer(c_size_t), parameter :: n=3
   real(c_double), dimension(n) :: tab=(/9.0, 10.0, 11.0/)
   real(c_double) :: variance

   interface
     function gsl_stats_variance(data,stride,n) &
                  & bind(c, name="gsl_stats_variance")
       use, intrinsic:: iso_c_binding
       implicit none
       real(c_double) :: gsl_stats_variance
       real(c_double), intent(in) :: data(*)
       integer(c_size_t), value :: stride, n
     end function gsl_stats_variance
   end interface

   variance = gsl_stats_variance(tab, 1_c_size_t, n)

   write(6,'(a,1x,f12.4)')"Variance:",variance

end program main
```

Back to initial problem

- `gsl_stats_variance` is a function
- `bind(c)` for interoperability
- Optional `name=` attribute for case sensitive C function name
- Return type is defined

gsl_stats_variance() with ISO_C_BINDING

gsl_stats_variance

```
double gsl_stats_variance(const double data[],
                          size_t stride, size_t n)
```

Fortran calling C

```
program main
   use, intrinsic:: iso_c_binding
   implicit none
   integer(c_size_t), parameter :: n=3
   real(c_double), dimension(n) :: tab=(/9.0, 10.0, 11.0/)
   real(c_double) :: variance

   interface
     function gsl_stats_variance(data,stride,n) &
                 & bind(c, name="gsl_stats_variance")
       use, intrinsic:: iso_c_binding
       implicit none
       real(c_double) :: gsl_stats_variance
       real(c_double), intent(in) :: data(*)
       integer(c_size_t), value :: stride, n
     end function gsl_stats_variance
   end interface

   variance = gsl_stats_variance(tab, 1_c_size_t, n)

   write(6,'(a,1x,f12.4)')"Variance:",variance

end program main
```

Back to initial problem

- new `c_size_t` type interoperable with C `size_t` type
- `value` attribute for `stride` and `n` arguments (sent by value!)

gsl_stats_variance() with ISO_C_BINDING

gsl_stats_variance

```
double gsl_stats_variance(const double data[],
                          size_t stride, size_t n)
```

Fortran calling C

```
program main
   use, intrinsic:: iso_c_binding
   implicit none
   integer(c_size_t), parameter :: n=3
   real(c_double), dimension(n) :: tab=(/9.0, 10.0, 11.0/)
   real(c_double) :: variance

   interface
     function gsl_stats_variance(data,stride,n) &
                 & bind(c, name="gsl_stats_variance")
       use, intrinsic:: iso_c_binding
       implicit none
       real(c_double) :: gsl_stats_variance
       real(c_double), intent(in) :: data(*)
       integer(c_size_t), value :: stride, n
     end function gsl_stats_variance
   end interface

   variance = gsl_stats_variance(tab, 1_c_size_t, n)

   write(6,'(a,1x,f12.4)')"Variance:",variance

end program main
```

Back to initial problem

- `intent(in)` for the array (`const` attribute in the C function)
- `data(*)` is assumed-size array interoperable with a C pointer/array
- Do not use `data(:)` here: an assumed-shape dummy argument is passed via a descriptor.

Modern Fortran approach

GNU Fortran 16.1.0

```
gfortran  main.f90 -o main.exe \
          -lgsl -lgslcblas -lm

./main.exe
Variance:         1.0000
```

NVIDIA Fortran 26.5-0

```
nvfortran  main.f90 -o main.exe \
           -lgsl -lgslcblas -lm

./main.exe
Variance:         1.0000
```

Intel Fortran 2026.1

```
ifx  main.f90 -o main.exe \
     -lgsl -lgslcblas -lm

bash-4.4$ ./main.exe
Variance:         1.0000
```

Cray Fortran 20.0

```
ftn main.f90 -o main.exe \
    -L../GSL-BIN/lib \
    -lgsl -lgslcblas -lm

./main.exe
Variance:         1.0000
```

gsl_stats_variance() with ISO_C_BINDING

Fortran calling C

```
program main
   use, intrinsic:: iso_c_binding
   implicit none
   integer(c_size_t), parameter :: n=6
   double precision, dimension(n) :: &
          &      tab=(/9.0, 0.0, 10.0, 0.0, 11.0, 0.0/)
   double precision :: variance

   interface
     function gsl_stats_variance(data,stride,n) &
                 & bind(c, name="gsl_stats_variance")
       use, intrinsic:: iso_c_binding
       implicit none
       real(c_double) :: gsl_stats_variance
       real(c_double), intent(in) :: data(*)
       integer(c_size_t), value :: stride, n
     end function gsl_stats_variance
   end interface

   variance = gsl_stats_variance(tab(1:6:2),&
                              & 1_c_size_t, 3_c_size_t)
   write(6,'(a,1x,f12.4)')"Variance sparse:",variance

end program main
```

gsl_stats_variance

```
double gsl_stats_variance(const double data[],
                          size_t stride, size_t n)
```

Back to initial problem

- `double precision` is interoperable with `real(c_double)`
- Stride in `tab` **must** works (Fortran 2018 specification)
- But `integer` is not interoperable with `integer(c_size_t)` (32 bits signed versus 64 bits unsigned)



What about calling Fortran from C?

C calling Fortran with ISO_C_BINDING

A simple Fortran module

```
module vector_operations

   implicit none

contains

   subroutine maxvalues(tab1, tab2, result, n)
      double precision, intent(in)  :: tab1(n), tab2(n)
      double precision, intent(out) :: result(n)
      integer, intent(in) :: n

      result = merge(tab1, tab2, tab1 > tab2)

   end subroutine maxvalues

end module vector_operations
```

Module goal

- One subroutine called maxvalues
- Receive `tab1` and `tab2` vectors and the size as input
- `result` contains the highest value of each elements at same index
- Uses `merge` Fortran intrinsic

C calling Fortran with ISO_C_BINDING

A simple Fortran program

```
subroutine show(info, val,n)
   implicit none
   character(len=*), intent(in) :: info
   double precision, intent(in) :: val(n)
   integer, intent(in) :: n

   write(*,'(A14,*(1x,f0.1))') info, val(:n)
end subroutine show

program main
  use vector_operations
  implicit none

  integer, parameter :: n=5
  double precision :: a(n) = [1.0, 8.0, 3.0, 7.0, 2.0]
  double precision :: b(n) = [4.0, 2.0, 9.0, 1.0, 5.0]
  double precision :: result(n)

  call  MaxValues(a, b, result, N);

  call show("Array a:",a,N);
  call show("Array b:",b,N);
  call show("Max values:",result,N);

end program main
```

Main application

- Subroutine `show` to print values
- Uses module `vector_operations`
- Fortran 2003 array constructors
- Call to `MaxValues(...)` from module `vector_operations`

C calling Fortran with ISO_C_BINDING

GNU Fortran 16.1.0

```
gfortran -c maxvalues_base.f90
gfortran -c main.f90
gfortran main.o maxvalues_base.o \
        -o mainf.exe
./mainf.exe
      Array a: 1.0 8.0 3.0 7.0 2.0
      Array b: 4.0 2.0 9.0 1.0 5.0
   Max values: 4.0 8.0 9.0 7.0 5.0
```

NVIDIA Fortran 26.5-0

```
nvfortran -c maxvalues_base.f90
nvfortran -c main.f90
nvfortran main.o maxvalues_base.o \
        -o mainf.exe
./mainf.exe
      Array a: 1.0 8.0 3.0 7.0 2.0
      Array b: 4.0 2.0 9.0 1.0 5.0
   Max values: 4.0 8.0 9.0 7.0 5.0
```



Intel Fortran 2026.1

```
ifx -c maxvalues_base.f90
ifx -c main.f90
ifx main.o maxvalues_base.o \
        -o mainf.exe
./mainf.exe
      Array a: 1.0 8.0 3.0 7.0 2.0
      Array b: 4.0 2.0 9.0 1.0 5.0
   Max values: 4.0 8.0 9.0 7.0 5.0
```

Cray Fortran 20.0

```
ftn -c maxvalues_base.f90
ftn -c main.f90
ftn main.o maxvalues_base.o \
        -o mainf.exe
./mainf.exe
      Array a: 1.0 8.0 3.0 7.0 2.0
      Array b: 4.0 2.0 9.0 1.0 5.0
   Max values: 4.0 8.0 9.0 7.0 5.0
```

C calling Fortran with ISO_C_BINDING

GNU Fortran 16.1.0

```
gfortran -c maxvalues_base.f90
gfortran -c main.f90
gfortran main.o maxvalues_base.o \
        -o mainf.exe
./mainf.exe
      Array a: 1.0 8.0 3.0 7.0 2.0
      Array b: 4.0 2.0 9.0 1.0 5.0
   Max values: 4.0 8
```

NVIDIA Fortran 26.5-0

```
nvfortran -c maxvalues_base.f90
nvfortran -c main.f90
nvfortran main.o maxvalues_base.o \
        -o mainf.exe
./mainf.exe
      Array a: 1.0 8.0 3.0 7.0 2.0
      Array b: 4.0 2.0 9.0 1.0 5.0
                .0 9.0 7.0 5.0
```

Intel Fortran 2026.1

```
ifx -c maxvalues_base
ifx -c main.f90
ifx main.o maxvalues_base.o \
        -o mainf.exe
./mainf.exe
      Array a: 1.0 8.0 3.0 7.0 2.0
      Array b: 4.0 2.0 9.0 1.0 5.0
   Max values: 4.0 8.0 9.0 7.0 5.0
```

Back to iso_c_binding

Let's use `maxvalues(...)` from our `vector_operations` module in a C application now!

```
se.f90
ftn -c main.f90
ftn main.o maxvalues_base.o \
        -o mainf.exe
./mainf.exe
      Array a: 1.0 8.0 3.0 7.0 2.0
      Array b: 4.0 2.0 9.0 1.0 5.0
   Max values: 4.0 8.0 9.0 7.0 5.0
```

C calling Fortran with ISO_C_BINDING

The C application

```c
#include <stdio.h>

void Show( char* info, const double val[], int n)
    printf("%-14.14s ",info);
    for (size_t i = 0; i < n; i++)
        printf("%g ", val[i]);
    printf("\n");
}

extern void MaxValues(const double a[],
                      const double b[],
                      double result[],
                      int n);

#define N 5
int main(void) {
    double a[N] = {1.0, 8.0, 3.0, 7.0, 2.0};
    double b[N] = {4.0, 2.0, 9.0, 1.0, 5.0};
    double result[N];

    MaxValues(a, b, result, N);

    Show("Array a:",a,N);
    Show("Array b:",b,N);
    Show("Max values:",result,N);
    return 0;
}
```

The C application

- Procedure `Show(...)` to print values
- `MaxValues` is `extern`
- Yes I like `PascalCase` notation in C!

Fortran maxvalue signature

```
subroutine maxvalues(tab1, tab2, result, n)
  double precision, intent(in) ::tab1(n)
  double precision, intent(in) ::tab2(n)
  double precision, intent(out)::result(n)
  integer, intent(in) :: n
```

C calling Fortran with ISO_C_BINDING

The C application

```c
#include <stdio.h>

void Show( char* info, const double val[], int n)
    printf("%-14.14s ",info);
    for (size_t i = 0; i < n; i++)
        printf("%g ", val[i]);
    printf("\n");
}

extern void MaxValues(const double a[],
                      const double b[],
                      double result[],
                      int n);

#define N 5
int main(void) {
    double a[N] = {1.0, 8.0, 3.0, 7.0, 2.0};
    double b[N] = {4.0, 2.0, 9.0, 1.0, 5.0};
    double result[N];

    MaxValues(a, b, result, N);

    Show("Array a:",a,N);
    Show("Array b:",b,N);
    Show("Max values:",result,N);
    return 0;
}
```

The C application

- Procedure `Show(...)` to print values
- `MaxValues` is `extern`
- Yes I like `PascalCase` notation in C!

Fortran maxvalue signature

```
subroutine maxvalues(tab1, tab2, result, n)
  double precision, intent(in) ::tab1(n)
  double precision, intent(in) ::tab2(n)
  double precision, intent(out)::result(n)
  integer, intent(in) :: n
```

Back to iso_c_binding

Let's implement corresponding `iso_c_binding` in `vector_operations` module now!

C calling Fortran with ISO_C_BINDING

ISO_C_BINDING in vector_operations module

```
module vector_operations

   use, intrinsic :: iso_c_binding
   implicit none

contains

   subroutine maxvalues(tab1, tab2, result, n) &
      bind(C, name="MaxValues")
      real(c_double), intent(in)  :: tab1(n))
      real(c_double), intent(in)  :: tab2(n)
      real(c_double), intent(out) :: result(n)
      integer(c_int), value       :: n

      result = merge(tab1, tab2, tab1 > tab2)

   end subroutine maxvalues

end module vector_operations
```

Modifications

- Uses the `ISO_C_BINDING` module
- Add the `BIND(C)` attribute and specify the C function name (case-sensitive)
- Replace `DOUBLE PRECISION` with `real(c_double)`
- Replace default `INTEGER` with `integer(c_int)`
- Add the `VALUE` attribute to `n` parameter

C calling Fortran with ISO_C_BINDING

ISO_C_BINDING in vector_operations module

```
module vector_operations

   use, intrinsic :: iso_c_binding
   implicit none

contains

   subroutine maxvalues(tab1, tab2, result, n) &
      bind(C, name="MaxValues")
      real(c_double), intent(in)  :: tab1(n))
      real(c_double), intent(in)  :: tab2(n)
      real(c_double), intent(out) :: result(n)
      integer(c_int), value       :: n

      result = merge(tab1, tab2, tab1 > tab2)

   end subroutine maxvalues

end module vector_operations
```

Modifications

- Uses the `ISO_C_BINDING` module
- Add the `BIND(C)` attribute and specify the C function name (case-sensitive)
- Replace `DOUBLE PRECISION` with `real(c_double)`
- Replace default `INTEGER` with `integer(c_int)`
- Add the `VALUE` attribute to `n` parameter

C calling Fortran with ISO_C_BINDING

ISO_C_BINDING in vector_operations module

```
module vector_operations

   use, intrinsic :: iso_c_binding
   implicit none

contains

   subroutine maxvalues(tab1, tab2, result, n) &
      bind(C, name="MaxValues")
      real(c_double), intent(in)  :: tab1(n))
      real(c_double), intent(in)  :: tab2(n)
      real(c_double), intent(out) :: result(n)
      integer(c_int), value       :: n

      result = merge(tab1, tab2, tab1 > tab2)

   end subroutine maxvalues

end module vector_operations
```

Modifications

- Uses the `ISO_C_BINDING` module
- Add the `BIND(C)` attribute and specify the C function name (case-sensitive)
- Replace `DOUBLE PRECISION` with `real(c_double)`
- Replace default `INTEGER` with `integer(c_int)`
- Add the `VALUE` attribute to `n` parameter

C calling Fortran with ISO_C_BINDING

ISO_C_BINDING in vector_operations module

```
module vector_operations

   use, intrinsic :: iso_c_binding
   implicit none

contains

   subroutine maxvalues(tab1, tab2, result, n) &
      bind(C, name="MaxValues")
      real(c_double), intent(in)  :: tab1(n))
      real(c_double), intent(in)  :: tab2(n)
      real(c_double), intent(out) :: result(n)
      integer(c_int), value       :: n

      result = merge(tab1, tab2, tab1 > tab2)

   end subroutine maxvalues

end module vector_operations
```

Modifications

- Uses the `ISO_C_BINDING` module
- Add the `BIND(C)` attribute and specify the C function name (case-sensitive)
- Replace `DOUBLE PRECISION` with `real(c_double)`
- Replace default `INTEGER` with `integer(c_int)`
- Add the `VALUE` attribute to `n` parameter

C calling Fortran with ISO_C_BINDING

GNU Fortran 16.1.0

```
gcc -c  main.c
gfortran -c  maxvalues.f90
gcc  -o main.exe main.o maxvalues.o
./main.exe
    Array a:        1 8 3 7 2
    Array b:        4 2 9 1 5
    Max values:     4 8 9 7 5
```

NVIDIA Fortran 26.5-0

```
nvc -c  main.c
nvfortran -c  maxvalues.f90
nvc  -o main.exe main.o maxvalues.o
./main.exe
    Array a:        1 8 3 7 2
    Array b:        4 2 9 1 5
    Max values:     4 8 9 7 5
```



Intel Fortran 2026.1

```
icx -c  main.c
ifx -c  maxvalues.f90
icx  -o main.exe main.o maxvalues.o
./main.exe
    Array a:        1 8 3 7 2
    Array b:        4 2 9 1 5
    Max values:     4 8 9 7 5
```

Cray Fortran 20.0

```
cc -c  main.c
ftn -c  maxvalues.f90
cc  -o main.exe main.o maxvalues.o
./main.exe
    Array a:        1 8 3 7 2
    Array b:        4 2 9 1 5
    Max values:     4 8 9 7 5
```

C calling Fortran with ISO_C_BINDING

GNU Fortran 16.1.0

```
gfortran -c maxvalues.f90
gfortran -c main.f90
gfortran main.o maxvalues.o \
        -o mainf.exe
./mainf.exe
      Array a: 1.0 8.0 3.0 7.0 2.0
      Array b: 4.0 2.0 9.0 1.0 5.0
   Max values: 4.0 8
```

NVIDIA Fortran 26.5-0

```
nvfortran -c maxvalues.f90
nvfortran -c main.f90
nvfortran main.o maxvalues.o \
        -o mainf.exe
./mainf.exe
      Array a: 1.0 8.0 3.0 7.0 2.0
      Array b: 4.0 2.0 9.0 1.0 5.0
   Max values: 3.0 9.0 7.0 5.0
```

Interoperability of MaxValue

Still working with Fortran main application (of course)!

Intel Fortran 2026.1

```
ifx -c maxvalues.f90
ifx -c main.f90
ifx main.o maxvalues.o \
        -o mainf.exe
./mainf.exe
      Array a: 1.0 8.0 3.0 7.0 2.0
      Array b: 4.0 2.0 9.0 1.0 5.0
   Max values: 4.0 8.0 9.0 7.0 5.0
```



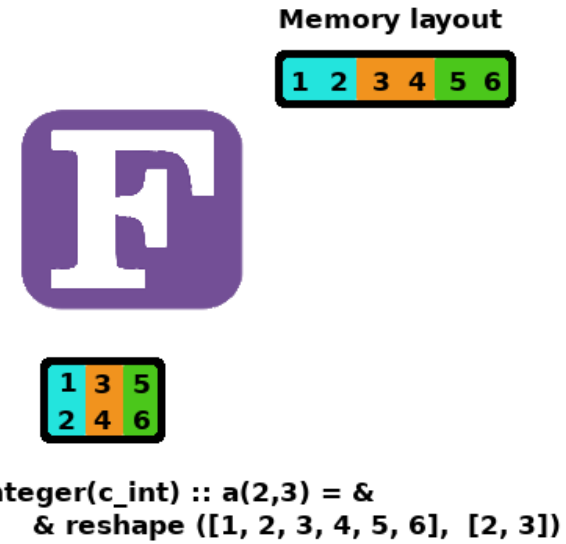
```
ftn -c maxvalues.f90
ftn -c main.f90
ftn main.o maxvalues.o \
        -o mainf.exe
./mainf.exe
      Array a: 1.0 8.0 3.0 7.0 2.0
      Array b: 4.0 2.0 9.0 1.0 5.0
   Max values: 4.0 8.0 9.0 7.0 5.0
```



But not all arrays are vectors ?

About multidimensionnal arrays

Memory layouts versus arrays shape



Arrays

- Arrays must be of interoperable type
- Fortran arrays are `Column_Major`
- C arrays are `Row_Major`
- C arrays indexes order must be swapped

About multidimensionnal arrays

Memory layouts versus arrays shape



Memory layout






```
int a[2][3] = {
  {1, 2, 3},
  {4, 5, 6}
  };
```





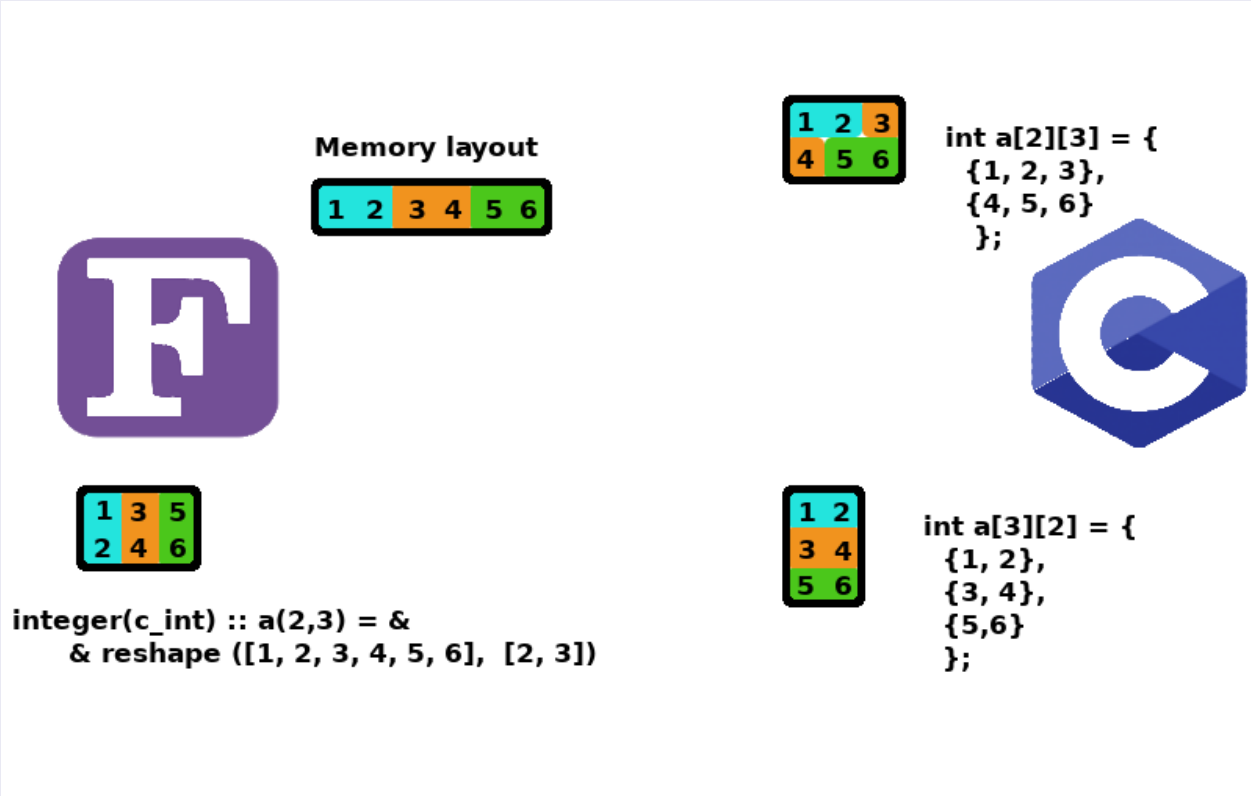

```
integer(c_int) :: a(2,3) = &
    & reshape ([1, 2, 3, 4, 5, 6],  [2, 3])
```

Arrays

- Arrays must be of interoperable type
- Fortran arrays are `Column_Major`
- C arrays are `Row_Major`
- C arrays indexes order must be swapped

About multidimensionnal arrays

Memory layouts versus arrays shape



Arrays

- Arrays must be of interoperable type
- Fortran arrays are `Column_Major`
- C arrays are `Row_Major`
- C arrays indexes order must be swapped

`integer(c_int) tab(18,3:7,4)` $\Longleftarrow$ *Fortran*

C $\Longrightarrow$ `int tab(4,5,18);`



What about derived types?

Interoperability of Fortran derived types

`Fortran type` $\Longleftrightarrow$ `C struct`

Requirements

For interoperability with `C`, **Fortran** derived types **must**:

- have the `bind(C)` attribute
- only interoperable components
- only interoperable parameters for the components
- have the same number of elements as in the C struct

Limitations

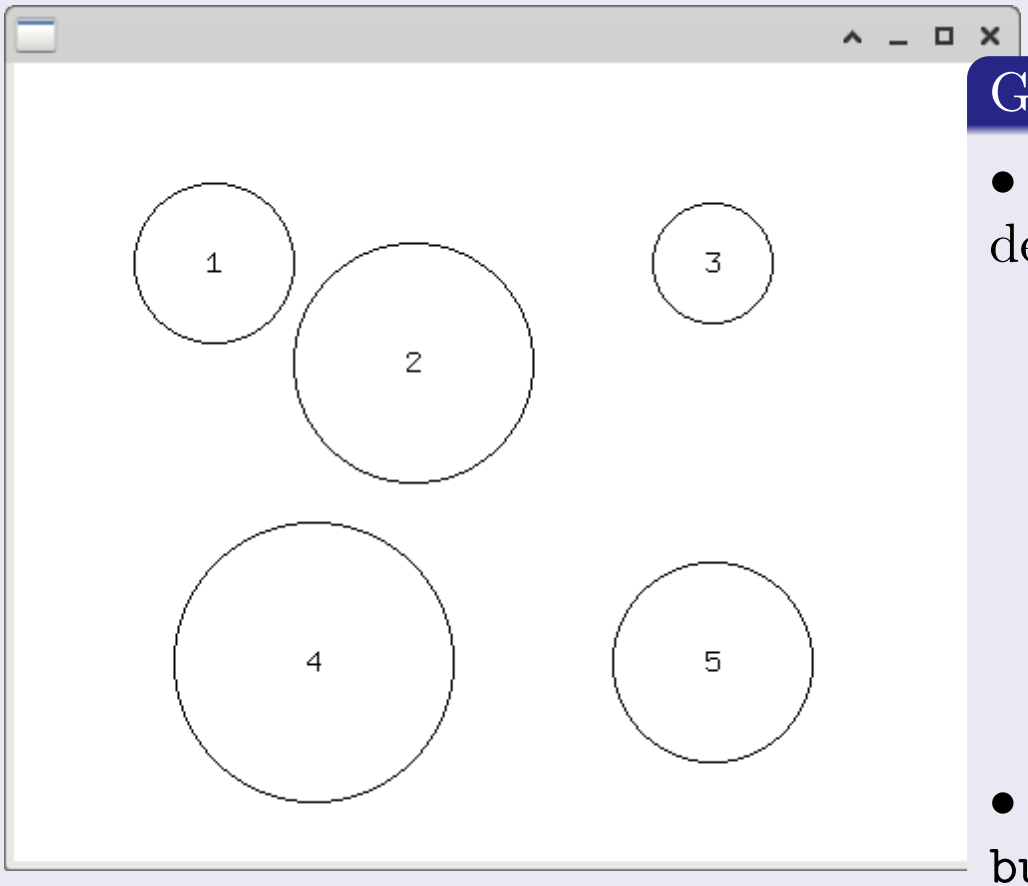
For interoperability with `C`, **Fortran** derived types **must not**:

- be a sequence
- have type parameters
- use the EXTENDS attribute
- have type-bound procedures
- have zero-sized arrays
- have `pointer` or `allocatable` components

Components correspond by position in the Fortran derived type and the C `struct`.

Interoperability of Fortran derived types

Plotting 2D *bubbles* from Fortran



Goal

• **[Fortran]** Create an interoperable derived type called `bubble` with:

- the x-coordinate of the center (component: `x`)
- the y-coordinate of the center (component: `y`)
- the radius (component: `radius`)

• **[Fortran]** Create an array of `bubble` objects

• **[C + libX11]** Draw the bubbles in an X11 window, showing their Fortran array indices

Interoperability of Fortran derived types

File *bubble_m.f90* (partial)

```
module bubble_m

  use, intrinsic :: iso_c_binding
  implicit none

  type, bind(C) :: bubble
     real(c_double) :: x
     real(c_double) :: y
     real(c_double) :: radius
  end type bubble

end module bubble_m
```

Interoperability

- `bind` attribute
- interoperable components
- interoperable parameters for the components

Goal

- **[Fortran]** Create an interoperable derived type called `bubble` with:
  - the x-coordinate of the center (component: `x`)
  - the y-coordinate of the center (component: `y`)
  - the radius (component: `radius`)

- **[Fortran]** Create an array of `bubble` objects

- **[C + libX11]** Draw the bubbles in an X11 window, showing their Fortran array indices

Interoperability of Fortran derived types

File *bubble_m.f90* (partial)

```
module bubble_m

  use, intrinsic :: iso_c_binding
  implicit none

  type, bind(C) :: bubble
     real(c_double) :: x
     real(c_double) :: y
     real(c_double) :: radius
  end type bubble

end module bubble_m
```

File *bubble.h* (partial)

```
#ifndef BUBBLE_H
#define BUBBLE_H

typedef struct {
    double x;
    double y;
    double radius;
} bubble;

#endif
```

Goal

- **[Fortran]** Create an interoperable derived type called `bubble` with:
  - the x-coordinate of the center (component: `x`)
  - the y-coordinate of the center (component: `y`)
  - the radius (component: `radius`)

- **[Fortran]** Create an array of `bubble` objects

- **[C + libX11]** Draw the bubbles in an X11 window, showing their Fortran array indices

Interoperability of Fortran derived types

File *main.f90* (partial)

```
1  program test_bubbles
2    use, intrinsic :: iso_c_binding
3    use bubble_m
4    implicit none
5
6    type(bubble) :: bubbles(5)
7
8    bubbles = [ &
9         bubble(100.0_c_double, 100.0_c_double, 40.0_c_double), &
10        bubble(200.0_c_double, 150.0_c_double, 60.0_c_double), &
11        bubble(350.0_c_double, 100.0_c_double, 30.0_c_double), &
12        bubble(150.0_c_double, 300.0_c_double, 70.0_c_double), &
13        bubble(350.0_c_double, 300.0_c_double, 50.0_c_double) ]
14
15 end program test_bubbles
```

Implementation

- use module `bubble_m` (3)
- create the array bubbles (6)
- initialize the array with an F 2003 `array constructor` (8-13)

- **[Fortran]** Create an array of `bubble` objects

- **[C + libX11]** Draw the bubbles in an X11 window, showing their Fortran array indices

Interoperability of Fortran derived types

File *show.c* (partial)

```c
#include <X11/Xlib.h>
#include <stdio.h>
#include <string.h>
#include "bubble.h"
void display_bubbles(const bubble bubbles[], size_t nbb)
{
  Display *display;
  Window window;
  GC gc;
  XEvent event;

  if (!init_graphics(&display, &window, &gc)) return;

  while (1) {
    XNextEvent(display, &event);
    if (event.type == Expose) {

      /* draw the bubbles! */

    XFlush(display);
    }
    if (event.type == KeyPress) break;
  }

  end_graphics(display, window, gc);
}
```

ate an interoperable
ed `bubble` with:

inate of the center
: `x`)
inate of the center
: `y`)

: `radius`)

ate an array of

- **[C + libX11]** Draw the bubbles in an X11 window, showing their Fortran array indices

Interoperability of Fortran derived types

File *show.c* (partial)

```c
while (1) {
  XNextEvent(display, &event);
  if (event.type == Expose) {
    for (size_t i = 0; i < nbb; i++) {
      /* Draw the bubble */
      int x = (int)(bubbles[i].x - bubbles[i].radius);
      int y = (int)(bubbles[i].y - bubbles[i].radius);
      int d = (int)(2.0 * bubbles[i].radius);
      XDrawArc(display, window, gc,
               x, y, d, d,
               0, 360 * 64);

      /* write the Fortran array index in the center */
      char label[32];
      snprintf(label, sizeof(label), "%zu", i + 1);
      XDrawString(display, window, gc,
                  (int)bubbles[i].x - 5,
                  (int)bubbles[i].y + 6,
                  label, strlen(label));
    }
    XFlush(display);
  }
  if (event.type == KeyPress) break;
}
```

interoperable
ble with:
f the center
f the center
us)
array of

- **[C + libX11]** Draw the bubbles in an X11 window, showing their Fortran array indices

Interoperability of Fortran derived types

File *bubble_m.f90*

```
module bubble_m

  use, intrinsic :: iso_c_binding
  implicit none

  type, bind(C) :: bubble
     real(c_double) :: x
     real(c_double) :: y
     real(c_double) :: radius
  end type bubble

  interface
     subroutine display_bubbles(bubbles, n) bind(C)
        import :: bubble, c_size_t
        type(bubble), intent(in) :: bubbles(*)
        integer(c_size_t), value :: n
     end subroutine display_bubbles
  end interface

end module bubble_m
```

`iso_c_binding`

- implement the Fortran interface for the `display_bubbles` C function in the module `bubble_m`
  - bind(C)
  - import definitions
  - assumed-size array with `bubbles(*)`
  - `value` attibute for `n`/`nbb` (matching name not requiered)

`display_bubbles` C function signature

```
void display_bubbles(const bubble bubbles[], size_t nbb)
```

Interoperability of Fortran derived types

File *main.f90*

```
1  program test_bubbles
2    use, intrinsic :: iso_c_binding
3    use bubble_m
4    implicit none
5
6    type(bubble) :: bubbles(5)
7
8    bubbles = [ &
9         bubble(100.0_c_double, 100.0_c_double, 40.0_c_double), &
10        bubble(200.0_c_double, 150.0_c_double, 60.0_c_double), &
11        bubble(350.0_c_double, 100.0_c_double, 30.0_c_double), &
12        bubble(150.0_c_double, 300.0_c_double, 70.0_c_double), &
13        bubble(350.0_c_double, 300.0_c_double, 50.0_c_double) ]
14
15   call display_bubbles(bubbles, size(bubbles, 1, kind=c_size_t))
16
17 end program test_bubbles
```

Implementation

- use module `bubble_m` with `display_bubbles` function interface
- create the array bubbles
- initialize the array with an F 2003 `array constructor`
- call the C function to draw the bubbles from the array.

Interoperability of Fortran derived types

GNU Fortran 16.1.0

```
gfortran -c bubble_m.f90
gfortran -c main.f90
gcc -c show.c
gfortran main.o bubble_m.o show.o \
            -o bubbles.exe  -lX11
./bubbles.exe
```

NVIDIA Fortran 26.5-0

```
nvfortran -c bubble_m.f90
nvfortran -c main.f90
nvc -c show.c
nvfortran main.o bubble_m.o show.o \
               -o bubbles.exe  -lX11
./bubbles.exe
```

Intel Fortran 2026.1

```
ifx -c bubble_m.f90
ifx -c main.f90
icx -c show.c
ifx main.o bubble_m.o show.o \
       -o bubbles.exe  -lX11
./bubbles.exe
```

Cray Fortran 20.0

```
ftn -c bubble_m.f90
ftn -c main.f90
cc -c show.c
ftn main.o bubble_m.o show.o \
       -o bubbles.exe  -lX11
./bubbles.exe
```

Interoperability of Fortran derived types

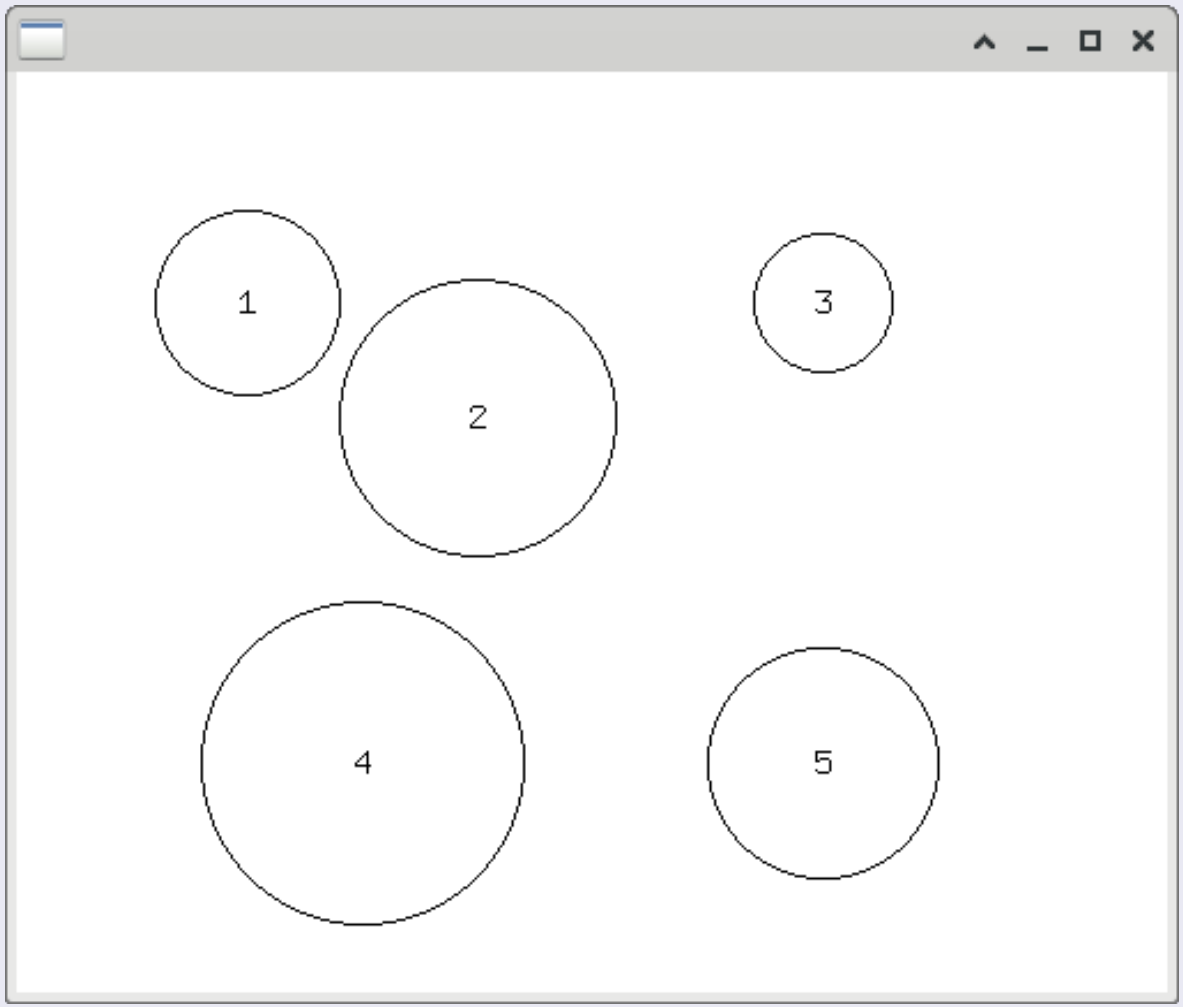
GNU Fortran 16.1.0

```
gfortran -c bubble_m.f90
gfortran -c main.f90
gcc -c show.c
gfortran main.o
            -o
./bubbles.exe
```

NVIDIA Fortran 26.5-0

```
nvfortran -c bubble_m.f90
nvfortran -c main.f90
```

```
_m.o show.o \
.exe  -lX11
```



Intel Fortran 20

```
ifx -c bubble_m
ifx -c main.f90
icx -c show.c
ifx main.o bubb
       -o bubbl
./bubbles.exe
```

```
how.o \
-lX11
```

Annexes

init_graphics(...) function

```
int init_graphics(Display **display, Window *window, GC *gc)
{
    *display = XOpenDisplay(NULL);

    if (*display == NULL) {
        fprintf(stderr, "Cannot open X display\n");
        return 0;
    }

    int screen = DefaultScreen(*display);

    *window = XCreateSimpleWindow(
        *display,
        RootWindow(*display, screen),
        0, 0, 500, 400,
        1,
        BlackPixel(*display, screen),
        WhitePixel(*display, screen));

    XSelectInput(*display, *window, ExposureMask | KeyPressMask);
    XMapWindow(*display, *window);

    *gc = XCreateGC(*display, *window, 0, NULL);

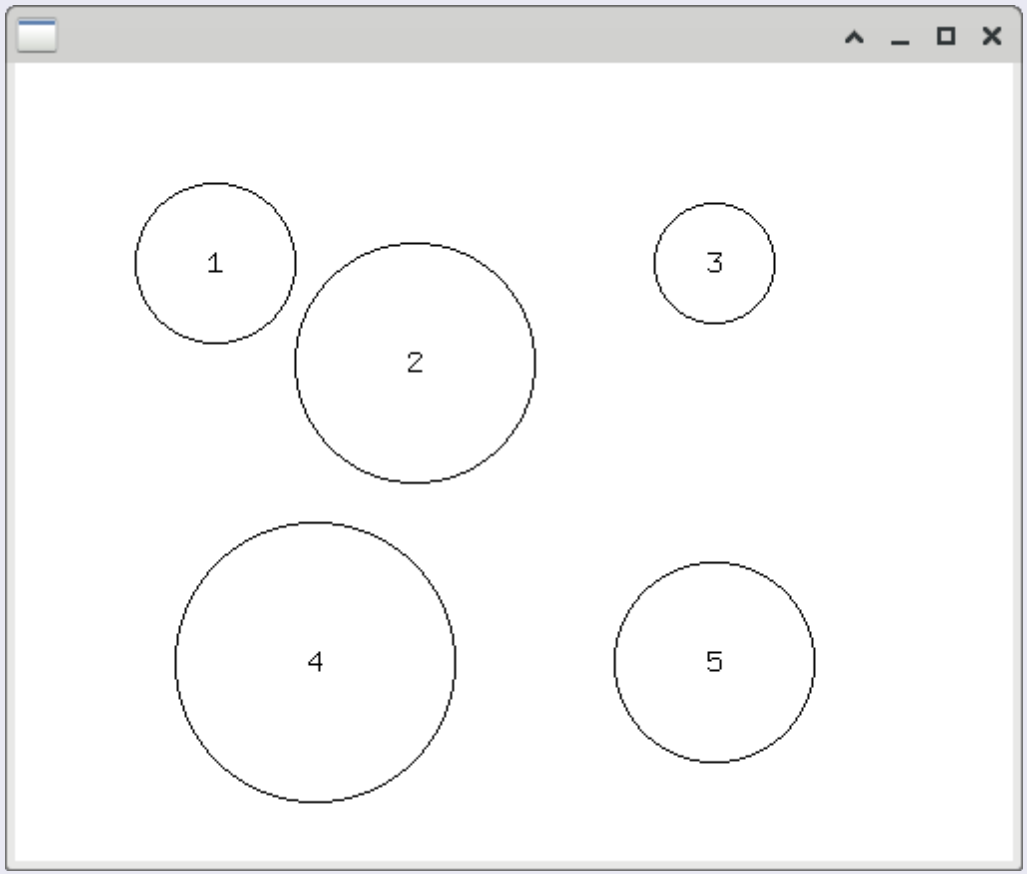
    return 1;
}
```

Annexes

end_graphics(...) function

```
void end_graphics(Display *display, Window window, GC gc)
{
    XFreeGC(display, gc);
    XDestroyWindow(display, window);
    XCloseDisplay(display);
}
```

Plotting 2D *bubbles* from Fortran





And C versus Fortran pointers ?

Interoperability of Fortran and C pointers

`Fortran pointer`	$\Longleftrightarrow$	`C pointer`
An opaque Fortran derived type		A memory address

ISO_C_BINDING derived types

- `type(c_ptr)` is interoperable with a **C object pointer**
- `type(c_funptr)` is interoperable with a **C function pointer**

ISO_C_BINDING named constants

- `c_null_ptr` null value for `c_ptr`
- `c_null_funptr` is the null value for `c_funptr`

ISO_C_BINDING functions

- `c_loc(x)` returns a `c_ptr` containing the C address of a Fortran object with the `TARGET` or `POINTER` attribute
- `c_funloc(proc)` returns in a `c_funptr` the C address of a Fortran procedure
- `c_f_pointer(cptr,fptr)` where `cptr` is a `c_ptr` with `intent(in)` and `fptr` is a Fortran `pointer` with `intent(out)`
- `c_f_procpointer(cptr,fptr)` where `cptr` is a `c_funptr` with `intent(in)` and `fptr` is a Fortran procedure `pointer` with `intent(out)`

Interoperability of Fortran and C pointers

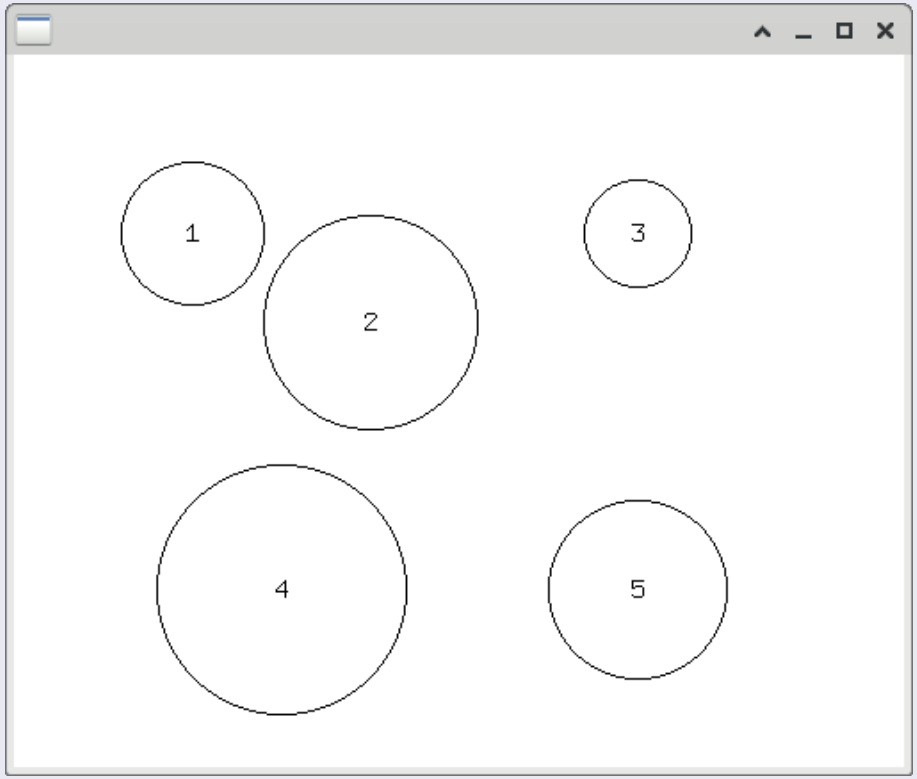
Ordering bubbles

- Goal: sort the bubble array before plotting:
  increase item index with bubble size.
- Fortran has no standard sorting procedure
- A generic `qsort` function is available in the C standard library

C qsort() function

```
void qsort(void *base,
           size_t nmemb,
           size_t size,
           int (*compar)(const void *,
                         const void *)
          );
```

Unsorted 2D bubbles plot from Fortran



Interoperability of Fortran and C pointers

Interface for qsort(...)

- Step 1: create a new module with an interface block with ISO_C_BINDING for the C `qsort(...)` stdlib function
- `qsort(...)` must remains a generic call as C `void*` arguments allow it in C language.

File *qsort_m.f90*

```
module qsort_m
  implicit none
  interface
    subroutine qsort(base,nelem,elemsize,comparison)&
                    & bind(C, name="qsort")
      use, intrinsic :: iso_c_binding, &
          & only: c_ptr, c_funptr, c_size_t

      type(c_ptr), value          :: base
      integer(c_size_t), value    :: nelem
      integer(c_size_t), value    :: elemsize
      type(c_funptr), value       :: comparison
    end subroutine qsort
  end interface
end module qsort_m
```

C qsort() function

```
void qsort(void *base,
           size_t nmemb,
           size_t size,
           int (*compar)(const void *, const void *)
          );
```

Interoperability of Fortran and C pointers

Interface for qsort(...)

- Subroutine is "bind(C)"
- C fonction name is specified
- Only required symbols are imported from `iso_c_binding`: `c_ptr`, `c_funptr` and `c_size_t`

File *qsort_m.f90*

```
module qsort_m
  implicit none
  interface
    subroutine qsort(base,nelem,elemsize,comparison)&
                        & bind(C, name="qsort")
      use, intrinsic :: iso_c_binding, &
          & only: c_ptr, c_funptr, c_size_t

      type(c_ptr), value         :: base
      integer(c_size_t), value   :: nelem
      integer(c_size_t), value   :: elemsize
      type(c_funptr), value      :: comparison
    end subroutine qsort
  end interface
end module qsort_m
```

C qsort() function

```
void qsort(void *base,
           size_t nmemb,
           size_t size,
           int (*compar)(const void *, const void *)
          );
```

Interoperability of Fortran and C pointers

Interface for qsort

- `base` is a `void*` pointer on the first element of the array
- `base` will be a `c_ptr`
- **WARNING:** `qsort(...)` needs the **value** of this pointer (the memory address)

File *qsort_m.f90*

```
module qsort_m
  implicit none
  interface
    subroutine qsort(base,nelem,elemsize,comparison)&
                     & bind(C, name="qsort")
      use, intrinsic :: iso_c_binding, &
          & only: c_ptr, c_funptr, c_size_t

      type(c_ptr), value          :: base
      integer(c_size_t), value    :: nelem
      integer(c_size_t), value    :: elemsize
      type(c_funptr), value       :: comparison
    end subroutine qsort
  end interface
end module qsort_m
```

C qsort() function

```
void qsort(void *base,
           size_t nmemb,
           size_t size,
           int (*compar)(const void *, const void *)
          );
```

Interoperability of Fortran and C pointers

Interface for qsort

- `nmemb` and `size` are `size_t` scalars
- `nelem` and `elemsize` will be `c_size_t` interoperable integers
- **WARNING:** `qsort(...)` needs the **value** of these arguments

File *qsort_m.f90*

```
module qsort_m
  implicit none
  interface
    subroutine qsort(base,nelem,elemsize,comparison)&
                     & bind(C, name="qsort")
      use, intrinsic :: iso_c_binding, &
          & only: c_ptr, c_funptr, c_size_t

      type(c_ptr), value          :: base
      integer(c_size_t), value    :: nelem
      integer(c_size_t), value    :: elemsize
      type(c_funptr), value       :: comparison
    end subroutine qsort
  end interface
end module qsort_m
```

C qsort() function

```
void qsort(void *base,
           size_t nmemb,
           size_t size,
           int (*compar)(const void *, const void *)
          );
```

Interoperability of Fortran and C pointers

Interface for qsort

- `compar` is a pointer to a function, comparison will be a `c_funptr`
- **WARNING:** `qsort(...)` needs the **value** of the function pointer (the address)

File *qsort_m.f90*

```
module qsort_m
  implicit none
  interface
    subroutine qsort(base,nelem,elemsize,comparison)&
                     & bind(C, name="qsort")
      use, intrinsic :: iso_c_binding, &
          & only: c_ptr, c_funptr, c_size_t

      type(c_ptr), value         :: base
      integer(c_size_t), value   :: nelem
      integer(c_size_t), value   :: elemsize
      type(c_funptr), value      :: comparison
    end subroutine qsort
  end interface
end module qsort_m
```

C qsort() function

```
void qsort(void *base,
           size_t nmemb,
           size_t size,
           int (*compar)(const void *, const void *)
          );
```

Interoperability of Fortran and C pointers

compare_bub(...)

- Step 2: implement the compare_bub(...) function for both arguments
- Specific to the bubble type; place it in *bubble_m.f90*
- Must be callable from C

C qsort() function

```
void qsort(void *base,
           size_t nmemb,
           size_t size,
           int (*compar)(const void *, const void *));
```

File *bubble_m.f90* (partial)

```
module bubble_m
....
contains
   integer(c_int) function compare_bub(p1, p2) &
                  & bind(C)
      use, intrinsic :: iso_c_binding, &
           & only: c_ptr, c_f_pointer
      implicit none

      type(c_ptr), value :: p1, p2
      type(bubble), pointer :: a, b

      call c_f_pointer(p1, a)
      call c_f_pointer(p2, b)

      if (a%radius < b%radius) then
         compare_bub = -1
      else if (a%radius > b%radius) then
         compare_bub = 1
      else
         compare_bub = 0
      end if
   end function compare_bub
end module bubble_m
```

Interoperability of Fortran and C pointers

compare_bub(...)

compare_bub(...) :

- Returns an interoperable integer
- Accept 2 arguments
- Is bind(c)
- Use ISO_C_BINDING
- Import only required symbols

C qsort() function

```
void qsort(void *base,
           size_t nmemb,
           size_t size,
           int (*compar)(const void *, const void *));
```

File *bubble_m.f90* (partial)

```
module bubble_m
....
contains
   integer(c_int) function compare_bub(p1, p2) &
                  & bind(C)
      use, intrinsic :: iso_c_binding, &
           & only: c_ptr, c_f_pointer
      implicit none

      type(c_ptr), value :: p1, p2
      type(bubble), pointer :: a, b

      call c_f_pointer(p1, a)
      call c_f_pointer(p2, b)

      if (a%radius < b%radius) then
         compare_bub = -1
      else if (a%radius > b%radius) then
         compare_bub = 1
      else
         compare_bub = 0
      end if
   end function compare_bub
end module bubble_m
```

Interoperability of Fortran and C pointers

compare_bub(...)

- C side arguments are pointers `const void*` (addresses of immutable targets)
- Fortran arguments MUST be `c_ptr`
- C address is required, use `value` attribute again
- Immutable is lost on the target!

C qsort() function

```
void qsort(void *base,
           size_t nmemb,
           size_t size,
           int (*compar)(const void *, const void *));
```

File *bubble_m.f90* (partial)

```
module bubble_m
....
contains
   integer(c_int) function compare_bub(p1, p2) &
                  & bind(C)
      use, intrinsic :: iso_c_binding, &
           & only: c_ptr, c_f_pointer
      implicit none

      type(c_ptr), value :: p1, p2
      type(bubble), pointer :: a, b

      call c_f_pointer(p1, a)
      call c_f_pointer(p2, b)

      if (a%radius < b%radius) then
         compare_bub = -1
      else if (a%radius > b%radius) then
         compare_bub = 1
      else
         compare_bub = 0
      end if
   end function compare_bub
end module bubble_m
```

Interoperability of Fortran and C pointers

compare_bub(...)

- Fortran cannot access the target through a `c_ptr` directly
- Use `c_f_pointer(p1,a)` to associate `a` with the target at `p1`.
- Warning: type compatibility is not checked.

File *bubble_m.f90* (partial)

```
module bubble_m
....
contains
   integer(c_int) function compare_bub(p1, p2) &
                & bind(C)
      use, intrinsic :: iso_c_binding, &
           & only: c_ptr, c_f_pointer
      implicit none

      type(c_ptr), value :: p1, p2
      type(bubble), pointer :: a, b

      call c_f_pointer(p1, a)
      call c_f_pointer(p2, b)

      if (a%radius < b%radius) then
         compare_bub = -1
      else if (a%radius > b%radius) then
         compare_bub = 1
      else
         compare_bub = 0
      end if
   end function compare_bub
end module bubble_m
```

C qsort() function

```
void qsort(void *base,
           size_t nmemb,
           size_t size,
           int (*compar)(const void *, const void *));
```

Interoperability of Fortran and C pointers

compare_bub(...)

- Finally, a and b point to objects of type bubble
- Compare their radius components
- Return an integer(c_int) value: -1, 0 or 1

Done for the ordering function

C qsort() function

```
void qsort(void *base,
           size_t nmemb,
           size_t size,
           int (*compar)(const void *, const void *));
```

File *bubble_m.f90* (partial)

```
module bubble_m
....
contains
   integer(c_int) function compare_bub(p1, p2) &
                  & bind(C)
      use, intrinsic :: iso_c_binding, &
           & only: c_ptr, c_f_pointer
      implicit none

      type(c_ptr), value :: p1, p2
      type(bubble), pointer :: a, b

      call c_f_pointer(p1, a)
      call c_f_pointer(p2, b)

      if (a%radius < b%radius) then
         compare_bub = -1
      else if (a%radius > b%radius) then
         compare_bub = 1
      else
         compare_bub = 0
      end if
   end function compare_bub
end module bubble_m
```

Interoperability of Fortran and C pointers

Calling qsort(...) from Fortran

- c_ptr is needed for the first element in the array bubbles
- Use c_loc(...) to get the address
- WARNING: for c_loc(XX), XX must be a fortran pointer or have the target attribute.

File *main.f90*

```
program test_bubbles
  use, intrinsic :: iso_c_binding
  use bubble_m
  implicit none

  type(bubble), target :: bubbles(5)

  bubbles = [ &
       bubble(100.0_c_double, 100.0_c_double, 40.0_c_double), &
       bubble(200.0_c_double, 150.0_c_double, 60.0_c_double), &
       bubble(350.0_c_double, 100.0_c_double, 30.0_c_double), &
       bubble(150.0_c_double, 300.0_c_double, 70.0_c_double), &
       bubble(350.0_c_double, 300.0_c_double, 50.0_c_double) ]

  call qsort( c_loc(bubbles(1)),                  &
              size(bubbles, kind=c_size_t),       &
              c_sizeof(bubbles(1)),               &
              c_funloc(compare_bub) )

  call display_bubbles(bubbles, size(bubbles, 1, kind=c_size_t))

end program test_bubbles
```

Interoperability of Fortran and C pointers

File *main.f90*

```
program test_bubbles
  use, intrinsic :: iso_c_binding
  use bubble_m
  implicit none

  type(bubble), target :: bubbles(5)

  bubbles = [ &
       bubble(100.0_c_double, 100.0_c_double, 40.0_c_double), &
       bubble(200.0_c_double, 150.0_c_double, 60.0_c_double), &
       bubble(350.0_c_double, 100.0_c_double, 30.0_c_double), &
       bubble(150.0_c_double, 300.0_c_double, 70.0_c_double), &
       bubble(350.0_c_double, 300.0_c_double, 50.0_c_double) ]

  call qsort( c_loc(bubbles(1)),                  &
              size(bubbles, kind=c_size_t),       &
              c_sizeof(bubbles(1)),               &
              c_funloc(compare_bub) )

  call display_bubbles(bubbles, size(bubbles, 1, kind=c_size_t))

end program test_bubbles
```

Calling `qsort(...)` from Fortran

- Provides the number of elements in the array `bubbles`
- Use Fortran intrinsic `size(...)`
- Use optionnal `kind=c_size_t` argument to return a required `c_size_t`

Interoperability of Fortran and C pointers

Calling qsort(...) from Fortran

- Provides the size **in bytes** of one element of the array `bubbles`
- Use Fortran intrinsic `c_sizeof(...)` from the `iso_c_binding` module
- `c_sizeof(...)` returns a required `c_size_t`

File *main.f90*

```
program test_bubbles
  use, intrinsic :: iso_c_binding
  use bubble_m
  implicit none

  type(bubble), target :: bubbles(5)

  bubbles = [ &
       bubble(100.0_c_double, 100.0_c_double, 40.0_c_double), &
       bubble(200.0_c_double, 150.0_c_double, 60.0_c_double), &
       bubble(350.0_c_double, 100.0_c_double, 30.0_c_double), &
       bubble(150.0_c_double, 300.0_c_double, 70.0_c_double), &
       bubble(350.0_c_double, 300.0_c_double, 50.0_c_double) ]

  call qsort( c_loc(bubbles(1)),                  &
              size(bubbles, kind=c_size_t),       &
              c_sizeof(bubbles(1)),               &
              c_funloc(compare_bub) )

  call display_bubbles(bubbles, size(bubbles, 1, kind=c_size_t))

end program test_bubbles
```

Interoperability of Fortran and C pointers

File *main.f90*

```
program test_bubbles
  use, intrinsic :: iso_c_binding
  use bubble_m
  implicit none

  type(bubble), target :: bubbles(5)

  bubbles = [ &
       bubble(100.0_c_double, 100.0_c
       bubble(200.0_c_double, 150.0_c
       bubble(350.0_c_double, 100.0_c_double, 30.0_c_double), &
       bubble(150.0_c_double, 300.0_c_double, 70.0_c_double), &
       bubble(350.0_c_double, 300.0_c_double, 50.0_c_double) ]

  call qsort( c_loc(bubbles(1)),                  &
              size(bubbles, kind=c_size_t),       &
              c_sizeof(bubbles(1)),               &
              c_funloc(compare_bub) )

  call display_bubbles(bubbles, size(bubbles, 1, kind=c_size_t))

end program test_bubbles
```

Calling qsort(...) from Fortran

- Passes a pointer to the C-interoperable **compare_bub(...)** function defined in module `bubble_m`
- Use the Fortran intrinsic `c_funloc(...)` from the `iso_c_binding` module
- `c_funloc(...)` returns the `c_funptr` required by `qsort`

Interoperability of Fortran and C pointers

GNU Fortran 16.1.0

```
gfortran -c bubble_m.f90 \
         -o bubble_m.o
gfortran -c qsort_m.f90 -o qsort_m.o
gfortran -c main.f90 -o main.o
gcc -c show.c -o show.o
gfortran main.o bubble_m.o show.o \
    qsort_m.o -o bubbles.exe  -lX11
./bubbles.exe
```

NVIDIA Fortran 26.5-0

```
nvfortran -c bubble_m.f90 \
          -o bubble_m.o
nvfortran -c qsort_m.f90 -o qsort_m.o
nvfortran -c main.f90 -o main.o
nvc -c show.c -o show.o
nvfortran main.o bubble_m.o show.o \
    qsort_m.o -o bubbles.exe  -lX11
./bubbles.exe
```

Intel Fortran 2026.1

```
ifx -c bubble_m.f90 -o bubble_m.o
ifx -c qsort_m.f90 -o qsort_m.o
ifx -c main.f90 -o main.o
icx -c show.c -o show.o
ifx main.o bubble_m.o show.o \
    qsort_m.o -o bubbles.exe  -lX11
./bubbles.exe
```

Cray Fortran 20.0

```
ftn -c bubble_m.f90 -o bubble_m.o
ftn -c qsort_m.f90 -o qsort_m.o
ftn -c main.f90 -o main.o
cc -c show.c -o show.o
ftn main.o bubble_m.o show.o \
    qsort_m.o -o bubbles.exe  -lX11
./bubbles.exe
```

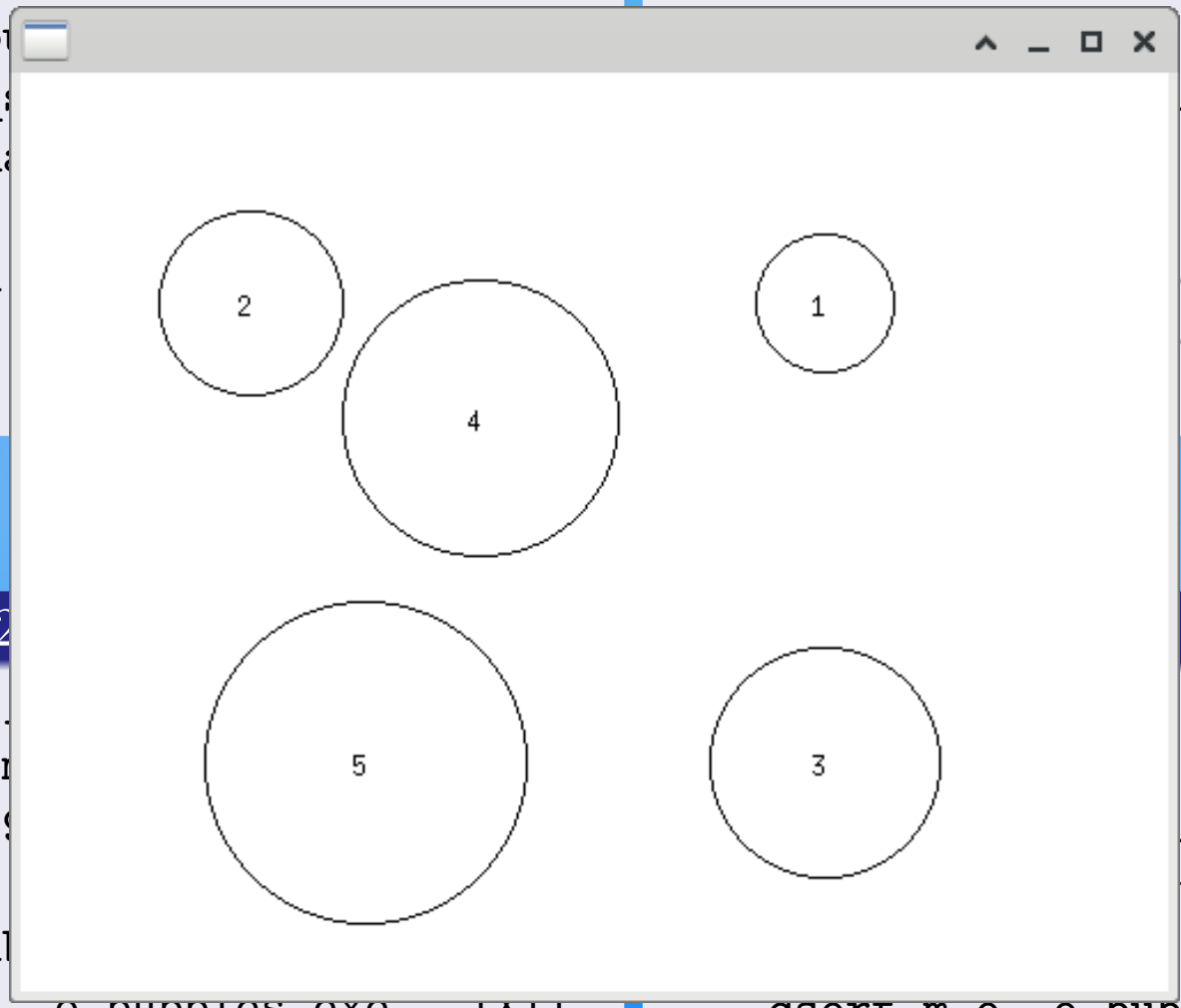
Interoperability of Fortran and C pointers

GNU Fortran 16.1.0

```
gfortran -c bubble_m.f90 \
         -o bu
gfortran -c qs
gfortran -c ma
gcc -c show.c
gfortran main
    qsort_m.o
./bubbles.exe
```

NVIDIA Fortran 26.5-0

```
nvfortran -c bubble_m.f90 \
m.o
.f90 -o qsort_m.o
0 -o main.o
w.o
ble_m.o show.o \
bles.exe  -lX11
```



Intel Fortran 2

```
ifx -c bubble
ifx -c qsort_
ifx -c main.f
icx -c show.c
ifx main.o bu
    qsort_m.o -o bubbles.exe  -lX11
./bubbles.exe
```

```
-o bubble_m.o
o qsort_m.o
ain.o
.o
o show.o \
    qsort_m.o -o bubbles.exe  -lX11
./bubbles.exe
```



What about the array descriptors mentioned slide 23?

Descriptors in ISO_C_BINDING

gsl_stats_variance

```
double gsl_stats_variance(const double data[],
                          size_t stride, size_t n)
```

Interface block

```
interface
    function gsl_stats_variance(data,stride,n) &
            & bind(c, name="gsl_stats_variance")
    use, intrinsic:: iso_c_binding
    implicit none
    real(c_double) :: gsl_stats_variance
    real(c_double), intent(in) :: data(*)
    integer(c_size_t), value :: stride, n
    end function gsl_stats_variance
end interface
```

"data" array declaration

- `intent(in)` for the array (`const` attribute in the C function)
- `data(*)` is **assumed-size array** interoperable with a C pointer/array
- **Do not use `data(:)` here: an assumed-shape dummy argument is passed via a descriptor.**

Remember!

Descriptors in ISO_C_BINDING

Push the limits!

Previous slides show that only Fontran/C very similar features can be interoperables.

Descriptors allow dummy arguments:

- assumed-shape array *(ex: tab(:,:))*
- assumed-rank array *(ex: tab(..))*
- allocatable
- pointer
- assumed-length character

ISO_Fortran_binding.h

`ISO_Fortran_binding.h` C header file:

- originated in ISO/IEC TS 29113:2012, which extended Fortran 2003/2008 interoperability,
- is part of the Fortran 2018 standard,
- is used in C code to manage descriptors.

`ISO_Fortran_binding.h` provides:

- `CFI_cdesc_t` type for descriptors,
- macro definitions,
- C procedure declarations.

Descriptors in ISO_C_BINDING

CFI_cdesc_t type [minimal] normalized attributes

```
void * base_addr;     // address of the data

size_t elem_len;      // For character: size in bytes of the string
                      // Otherwise: size in bytes of one element

CFI_attribute_t attribute; // - CFI_attribute_allocatable
                           // - CFI_attribute_pointer
                           // - CFI_attribute_other

CFI_rank_t rank;      // -> 0 for scalar
                      // -> 1 for vector
                      // -> 2 ....etc

CFI_type_t type;      // - CFI_type_int (for int)
                      // - CFI_type_size_t (for size_t)
                      // ...
                      // - CFI_type_long_double (for long double)
                      // ...
                      // -CFI_type_cptr (for C data pointer)
                      // -CFI_type_struct (for interoperable C structure)
                      // - CFI_type_other (for a non-interoperable type)

CFI_dim_t dim[];      // One CFI_dim_t for each dimension in Fortran order
                      //  -> lower_bound
                      //  -> extent: size of this dimension
                      //  -> sm: memory stride between elements in this dimension
```

Descriptors in ISO_C_BINDING

ISO_Fortran_binding.h API

```
// Deallocate an allocatable or pointer array
int CFI_deallocate (CFI_cdesc_t *desc);

// Allocate an allocatable or pointer array
int CFI_allocate (CFI_cdesc_t * desc,
                  const CFI_index_t lower_bounds[], // array of lower bounds
                  const CFI_index_t extents[],      // array of extents
                  size_t elem_len);
// Test whether the data are contiguous
int CFI_is_contiguous (const CFI_cdesc_t *);

...
```

Warning

Never manage memory allocation for `base_addr` directly

Descriptors in ISO_C_BINDING: improve bubble

Improve bubble_m module (*bubble_m.f90*)

```
module bubble_m

  use, intrinsic :: iso_c_binding
  implicit none

...

  interface
     subroutine display_bubbles(bubbles, n) bind(C)
        import :: bubble, c_size_t
        type(bubble), intent(in) :: bubbles(*)
        integer(c_size_t), value :: n
     end subroutine display_bubbles

     subroutine display_bubbles_desc(bubbles) bind(C)
        import :: bubble, c_size_t
        type(bubble), intent(in) :: bubbles(:)
     end subroutine display_bubbles_desc
  end interface

contains

...

end module bubble_m
```

display_bubbles_desc()

- New *display_bubbles_desc* C function
- Do not receive the array size
- Argument `bubbles(:)` sent as a descriptor
- `bubbles(:)` is an assumed-shape array

Descriptors in ISO_C_BINDING: improve bubble

Changes in main.f90

```
program test_bubbles
  use, intrinsic :: iso_c_binding
  use bubble_m
  use qsort_m
  implicit none

  type(bubble), target :: bubbles(5)

  bubbles = [ &
       bubble(100.0_c_double, 100.0_c_double, 40.0_c_double), &
       bubble(200.0_c_double, 150.0_c_double, 60.0_c_double), &
       bubble(350.0_c_double, 100.0_c_double, 30.0_c_double), &
       bubble(150.0_c_double, 300.0_c_double, 70.0_c_double), &
       bubble(350.0_c_double, 300.0_c_double, 50.0_c_double) ]

  call qsort( c_loc(bubbles(1)),                  &
              size(bubbles, kind=c_size_t),       &
              c_sizeof(bubbles(1)),               &
              c_funloc(compare_bub) )

  call display_bubbles_desc(bubbles)

end program test_bubbles
```

Use descriptor

- call to display_bubbles_desc()
- only one argument: the bubble array

Descriptors in ISO_C_BINDING: improve bubble

C implementation for display_bubbles_desc(...)

```c
#include <ISO_Fortran_binding.h>

/* Descriptor based version of display_bubbles() */
void display_bubbles_desc(const CFI_cdesc_t *mydesc)
   {
   const bubble *bubbles;
   size_t n;

   if (mydesc == NULL) return;

   if (mydesc->base_addr == NULL) return;
   if (mydesc->rank != 1) return;
   if (mydesc->type != CFI_type_struct) return;

   if (! CFI_is_contiguous(mydesc)) return;

   /* First element in the array */
   bubbles = (const bubble *) mydesc->base_addr;

   /* Number of elements along first dimension */
   n = (size_t)mydesc->dim[0].extent;

   /* Use previous function */
   display_bubbles(bubbles, n);
   }
```

display_bubbles_desc()

- include the *ISO_Fortran_binding.h* header file
- single argument is a pointer to a `CFI_cdesc_t` type

Descriptors in ISO_C_BINDING: improve bubble

C implementation for display_bubbles_desc(...)

```c
#include <ISO_Fortran_binding.h>

/* Descriptor based version of display_bubbles() */
void display_bubbles_desc(const CFI_cdesc_t *mydesc)
   {
   const bubble *bubbles;
   size_t n;

   if (mydesc == NULL) return;

   if (mydesc->base_addr == NULL) return;
   if (mydesc->rank != 1) return;
   if (mydesc->type != CFI_type_struct) return;

   if (! CFI_is_contiguous(mydesc)) return;

   /* First element in the array */
   bubbles = (const bubble *) mydesc->base_addr;

   /* Number of elements along first dimension */
   n = (size_t)mydesc->dim[0].extent;

   /* Use previous function */
   display_bubbles(bubbles, n);
   }
```

display_bubbles_desc()

- check the argument is a valid C pointer

Descriptors in ISO_C_BINDING: improve bubble

C implementation for display_bubbles_desc(...)

```c
#include <ISO_Fortran_binding.h>

/* Descriptor based version of display_bubbles() */
void display_bubbles_desc(const CFI_cdesc_t *mydesc)
   {
   const bubble *bubbles;
   size_t n;

   if (mydesc == NULL) return;

   if (mydesc->base_addr == NULL) return;
   if (mydesc->rank != 1) return;
   if (mydesc->type != CFI_type_struct) return;

   if (! CFI_is_contiguous(mydesc)) return;

   /* First element in the array */
   bubbles = (const bubble *) mydesc->base_addr;

   /* Number of elements along first dimension */
   n = (size_t)mydesc->dim[0].extent;

   /* Use previous function */
   display_bubbles(bubbles, n);
   }
```

display_bubbles_desc()

Check the descriptor provides what we expect:

- at least one bubble to draw
- a rank-1 array is expected
- the elements are derived-type objects

Descriptors in ISO_C_BINDING: improve bubble

C implementation for display_bubbles_desc(...)

```c
#include <ISO_Fortran_binding.h>

/* Descriptor based version of display_bubbles() */
void display_bubbles_desc(const CFI_cdesc_t *mydesc)
   {
   const bubble *bubbles;
   size_t n;

   if (mydesc == NULL) return;

   if (mydesc->base_addr == NULL) return;
   if (mydesc->rank != 1) return;
   if (mydesc->type != CFI_type_struct) return;

   if (! CFI_is_contiguous(mydesc)) return;

   /* First element in the array */
   bubbles = (const bubble *) mydesc->base_addr;

   /* Number of elements along first dimension */
   n = (size_t)mydesc->dim[0].extent;

   /* Use previous function */
   display_bubbles(bubbles, n);
   }
```

display_bubbles_desc()

- with assumed-shape array contiguity is not guaranteed

- use `mydesc->dim[0].sm` to get the memory stride if required

Descriptors in ISO_C_BINDING: improve bubble

C implementation for display_bubbles_desc(...)

```c
#include <ISO_Fortran_binding.h>

/* Descriptor based version of display_bubbles() */
void display_bubbles_desc(const CFI_cdesc_t *mydesc)
   {
   const bubble *bubbles;
   size_t n;

   if (mydesc == NULL) return;

   if (mydesc->base_addr == NULL) return;
   if (mydesc->rank != 1) return;
   if (mydesc->type != CFI_type_struct) return;

   if (! CFI_is_contiguous(mydesc)) return;

   /* First element in the array */
   bubbles = (const bubble *) mydesc->base_addr;

   /* Number of elements along first dimension */
   n = (size_t)mydesc->dim[0].extent;

   /* Use previous function */
   display_bubbles(bubbles, n);
   }
```

display_bubbles_desc()

From the descriptor:

- get the address of the first element as a `(const bubble *)`
- get the extent of this array

And call the previous version of the drawing function!

Interoperability of Fortran and C pointers

GNU Fortran 16.1.0

```
gfortran -c bubble_m.f90 \
         -o bubble_m.o
gfortran -c qsort_m.f90 -o qsort_m.o
gfortran -c main.f90 -o main.o
gcc -c show.c -o show.o
gfortran main.o bubble_m.o show.o \
    qsort_m.o -o bubbles.exe  -lX11
./bubbles.exe
```

NVIDIA Fortran 26.5-0

```
nvfortran -c bubble_m.f90 \
          -o bubble_m.o
nvfortran -c qsort_m.f90 -o qsort_m.o
nvfortran -c main.f90 -o main.o
nvc -c show.c -o show.o
nvfortran main.o bubble_m.o show.o \
    qsort_m.o -o bubbles.exe  -lX11
./bubbles.exe
```

Intel Fortran 2026.1

```
ifx -c bubble_m.f90 -o bubble_m.o
ifx -c qsort_m.f90 -o qsort_m.o
ifx -c main.f90 -o main.o
icx -c show.c -o show.o
ifx main.o bubble_m.o show.o \
    qsort_m.o -o bubbles.exe  -lX11
./bubbles.exe
```

Cray Fortran 20.0

```
ftn -c bubble_m.f90 -o bubble_m.o
ftn -c qsort_m.f90 -o qsort_m.o
ftn -c main.f90 -o main.o
cc -c show.c -o show.o
ftn main.o bubble_m.o show.o \
    qsort_m.o -o bubbles.exe  -lX11
./bubbles.exe
```

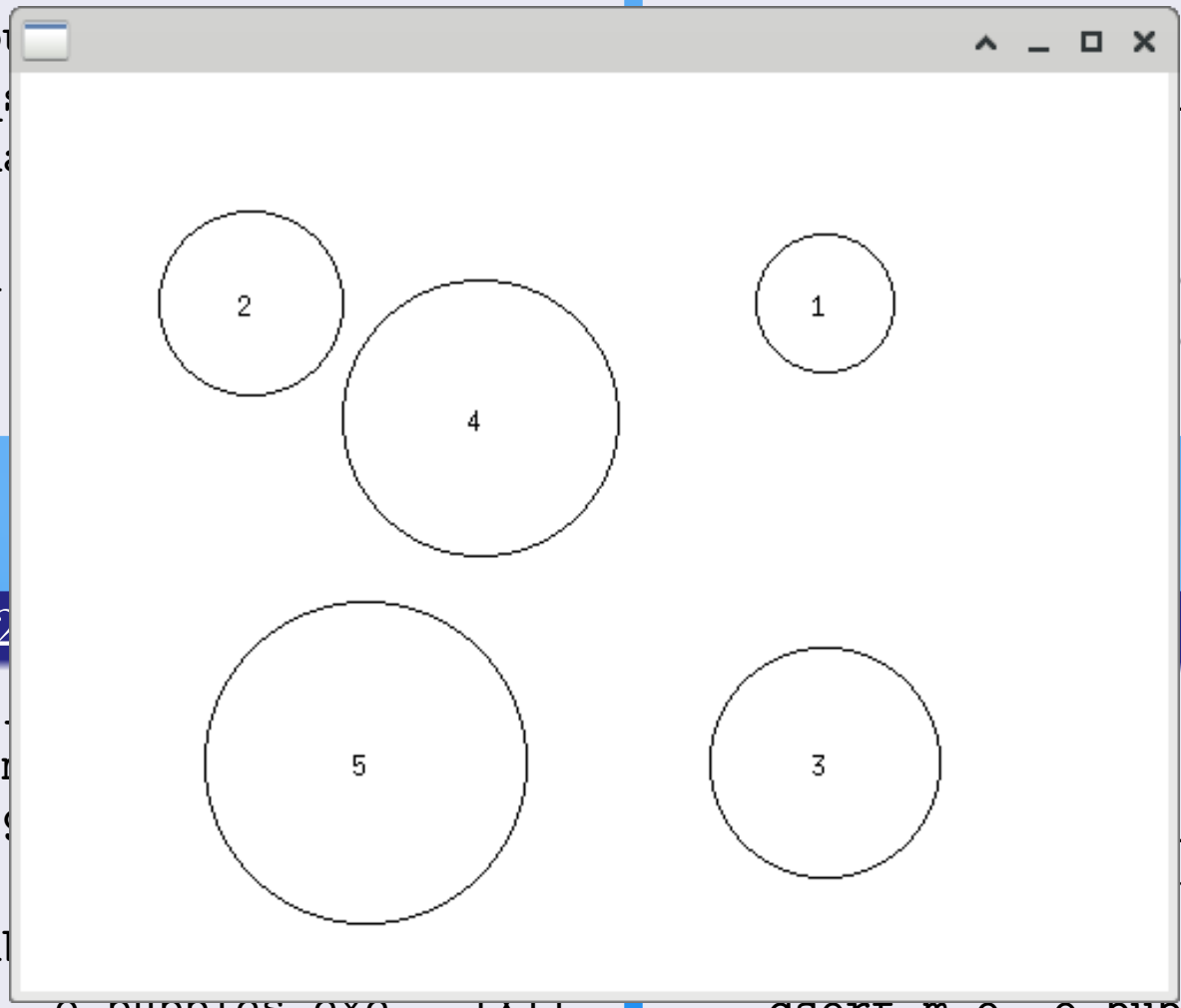
Interoperability of Fortran and C pointers

GNU Fortran 16.1.0

```
gfortran -c bubble_m.f90 \
         -o bu
gfortran -c qs
gfortran -c ma
gcc -c show.c
gfortran main
    qsort_m.o
./bubbles.exe
```

NVIDIA Fortran 26.5-0

```
nvfortran -c bubble_m.f90 \
m.o
.f90 -o qsort_m.o
0 -o main.o
w.o
ble_m.o show.o \
bles.exe  -lX11
```



Intel Fortran 2

```
ifx -c bubble
ifx -c qsort_
ifx -c main.f
icx -c show.c
ifx main.o bu
    qsort_m.o -o bubbles.exe  -lX11
./bubbles.exe
```

```
-o bubble_m.o
o qsort_m.o
ain.o
.o
o show.o \
    qsort_m.o -o bubbles.exe  -lX11
./bubbles.exe
```

Conclusion

ISO_C_BINDING

ISO_C_BINDING provides a robust, standardized, and portable way to interface Fortran and C in many situations

Topics not covered...

- multidimensional arrays (mentioned briefly)
- string handling
- ...

Limits of standard interoperability...

- runtime type checking for derived types across C and Fortran
- variadic C functions (e.g. `printf` or `scanf`)
- derived types with `ALLOCATABLE` or `POINTER` components
- ...