`Composite Datatypes in Fortran 95`

Aleksandar Donev

July 1, 2002

1 Testing fancy derived-type F2x facilities

This data-type describes a point embedded in Eucledian space of dimensionality *n_space_dims* which contains a datum of real numbers (physical observables) of length *n_dims*. For example, we could be measuring the potential (1 variable) over a grid of points in 3D.

1.1 Data points embedded in Eucledian space

"WEAVE.f90" 1.1 ≡

```
@m _DataPointType(n_dims_, n_space_dims_)

   _TYPE :: _ParameterizedType (Data_Point, n_dims_, n_space_dims_)
      _DeclareTypeParameters_NoInquiry(2, (n_dims, n_space_dims), (i_short, i_byte), n_dims_,
          n_space_dims_)
      REAL (KIND = r_wp), DIMENSION (n_space_dims_) :: coordinates = 0.0_r_wp
      REAL (KIND = r_wp), DIMENSION (n_dims_) :: data = 0.0_r_wp
   ENDTYPE

@m _DataPointsType(n_dims_, n_space_dims_)

   _TYPE :: _ParameterizedType (Data_Points, n_dims_, n_space_dims_)
      _DeclareTypeParameters(2, (n_dims, n_space_dims), (i_short, i_byte), n_dims_, n_space_dims_)
          // For quick parameter inquiry
      INTEGER :: n_points = 0    // The number of data points
      TYPE (_ParameterizedType(Data_Point, n_dims_, n_space_dims_)), DIMENSION (:),
           _DYNAMIC :: _NULLIFIED(points)    // The actual data
      _C_PTR :: coordinates_address = C_NULL_PTR
   ENDTYPE
```

"WEAVE.f90" 1.1.1 ≡

```
@m _CoordinateCentroid(n_dims_, n_space_dims_)
```

```
FUNCTION _ParameterizedProcedure(CoordinateCentroid, n_dims_,
      n_space_dims_)(data_points) RESULT(centroid)
  TYPE (_ParameterizedType(Data_Points, n_dims_, n_space_dims_)), INTENT (IN) :: data_points
  REAL (KIND = r_wp), DIMENSION (n_space_dims_) :: centroid

  INTEGER :: point

  centroid = 0.0_r_wp

    // Find the average along each coordinate axis
  DO point = 1, data_points % n_points
    centroid += data_points % points(point) % coordinates
  END DO

  centroid = centroid / data_points % n_points

END FUNCTION
```

1.2 Usage

```
"WEAVE.f90" 1.2 ≡
  @m _test_parameterized 0
```

```
@#if (_test_parameterized)

  MODULE Space_Embedded_Data
    USE Precision
    USE ISO_C_BINDING
    IMPLICIT NONE

    _DeclareParameterizedType(_DataPointType, 2, (n_dims, n_space_dims), 3, 2, (1, 2, 3), (2, 3))
    _DeclareParameterizedType(_DataPointsType, 2, (n_dims, n_space_dims), 3, 2, (1, 2, 3), (2, 3))

    _ParameterizedProcedureInterface(CoordinateCentroid, 2, (n_dims, n_space_dims), 3, 2, (1, 2, 3),
        (2, 3))

  CONTAINS

    _ParameterizedProcedureBody(_CoordinateCentroid, 2, (n_dims, n_space_dims), 3, 2, (1, 2, 3), (2, 3))

  END MODULE Space_Embedded_Data
@#endif

  PROGRAM Test_Datatypes
  USE Precision
  USE ISO_C_BINDING
  _USE_F2x
  USE Composite_Datatypes
  USE Space_Embedded_Data
  IMPLICIT NONE

@#if (_test_parameterized)
    // This is the old test-program for parametrized data-types:

  TYPE (_ParameterizedType(Data_Points, 1, 2)), TARGET :: potentials
```

```
   // Voltages (i.e. scalars) measured on a set of points in 2D
TYPE (_ParameterizedType(Data_Points, 3, 3)), TARGET :: displacements
   // A displacement/velocity field of a set of 3D points

REAL (KIND = r_wp), DIMENSION (:,:), POINTER :: coordinates
INTEGER :: coordinates_stride

INTEGER :: dim

potentials % n_points = 5
displacements % n_points = 3

ALLOCATE (potentials % points(potentials % n_points))
ALLOCATE (displacements % points(displacements % n_points))

WRITE (*, *)
WRITE (*, *) "The stride (in sizeof(r_wp)) between successive x coordinates is:",
    C_STRIDE(potentials % points(1) % coordinates(1), potentials % points(2) % coordinates(1))
WRITE (*, *) "The stride (in sizeof(r_wp)) between successive potentials is:",
    C_STRIDE(potentials % points(1) % data(1), potentials % points(2) % data(1))
WRITE (*, *) "The stride (in sizeof(r_wp)) between x coordinates and potential is:",
    C_STRIDE(potentials % points(1) % coordinates(1), potentials % points(1) % data(1))

WRITE (*, *)
WRITE (*, *) "The stride (in sizeof(r_wp)) between successive x coordinates is:",
    C_STRIDE(displacements % points(1) % coordinates(1), displacements % points(2) % coordinates(1))
WRITE (*, *) "The stride (in sizeof(r_wp)) between successive x displacements is:",
    C_STRIDE(displacements % points(1) % data(1), displacements % points(2) % data(1))
WRITE (*, *) "The stride (in sizeof(r_wp)) between x coordinates and x displacements
    is:", C_STRIDE(displacements % points(1) % coordinates(1), displacements % points(1) % data(1))

potentials % coordinates_address = C_LOC(potentials % points(1) % coordinates)
coordinates_stride = C_STRIDE(potentials % points(1) % coordinates(1), potentials % points(2) % coordinates(1))

CALL C_F_POINTER(CPTR = potentials % coordinates_address, FPTR = coordinates,
    SHAPE = (/ coordinates_stride, potentials % n_points /))
_AssignArrayPointer(2, coordinates, coordinates(1:potentials%n_space_dims, :), (1, _LBOUND(coordinates,
    i_wp, DIM = 1)))

DO dim = 1, potentials % n_space_dims
  CALL RANDOM_NUMBER(potentials % points(:) % coordinates(dim))
END DO

WRITE (*, *) "The centroid of the 2D data points is:", CoordinateCentroid(potentials)
WRITE (*, *) "--->Using the array pointer:", SUM(coordinates, DIM = 2) / potentials % n_points
```

@#else

```
   // This is the new test program for component pointer extraction:

_TYPE :: Hourly_Record
  REAL (KIND = r_wp) :: temperature(3) = 0.0      // Three different temperature readings
  LOGICAL (KIND = l_byte) :: synny = T
ENDTYPE

_TYPE :: Daily_Record
  TYPE (Hourly_Record), DIMENSION (24) :: hourly_records
  INTEGER (KIND = i_sp) :: sunrise = 7, sunset = 18
ENDTYPE

_TYPE :: Weekly_Record
```

```
      TYPE (Daily_Record), DIMENSION (7) :: daily_records
      REAL (KIND = r_sp) :: forecast_success(5)
   ENDTYPE

   INTEGER, PARAMETER :: n_x = 3, n_y = 3     // Intel can barely handle this size of grid!
   TYPE (Weekly_Record), DIMENSION (n_x, n_y), TARGET :: weekly_records     // Over a 2D grid

      // The reference we are after is:
      // weekly_records(:,:) % daily_records(:) % hourly_records(:) % temperature(:)

   TYPE (Component_Pointer_@&R_WP), DIMENSION(5) :: components
   REAL (KIND = r_wp), DIMENSION ( :, :, :, :, : ), POINTER :: temperatures

   REAL (KIND = r_wp), DIMENSION ( :, : ), ALLOCATABLE :: buffer
   REAL (KIND = r_wp), DIMENSION ( :, : ), POINTER :: selected_temperatures

   INTEGER :: i, j, day, hour

   DO i = 1, n_x
      DO j = 1, n_y
         DO day = 1, 7
            DO hour = 1, 24
               weekly_records(i, j) % daily_records(day) % hourly_records(hour) % temperature(:) =
                     1·10^6 * i + 1·10^4 * j + 1·10^3 * day + 1·10^1 * hour + (/ 1, 2, 3 /)
            END DO
         END DO
      END DO
   END DO

   components(1) % component ⇒ weekly_records(1, 1) % daily_records(1) % hourly_records(1) % temperature(:)
   components(2) % component ⇒ weekly_records(1, 1) % daily_records(1) % hourly_records(:) % temperature(1)
   components(3) % component ⇒ weekly_records(1, 1) % daily_records(:) % hourly_records(1) % temperature(1)
   components(4) % component ⇒ weekly_records(:, 1) % daily_records(1) % hourly_records(1) % temperature(1)
   components(5) % component ⇒ weekly_records(1, :) % daily_records(1) % hourly_records(1) % temperature(1)

   CALL ExtractComponent(components = components, component_pointer = temperatures)

   WRITE (*, *) "Lower bounds:", LBOUND(temperatures)
   WRITE (*, *) "Upper bounds:", UBOUND(temperatures)

      // The second temperature reading on Mondays and Wednesdays at 9:00 and 15:00 hours at grid
         point 3,1
   selected_temperatures ⇒ temperatures(2, 9:15:6, 1:3:2, 1, 3)

      // Now pack this into a buffer:
   ALLOCATE (buffer(SIZE(selected_temperatures, DIM = 1), SIZE(selected_temperatures, DIM = 2)))
   buffer = selected_temperatures     // Pack into contiguous memory
   WRITE (*, *) "The buffer contents:", INT(buffer)
   DEALLOCATE (buffer)
@#endif

   END PROGRAM Test_Datatypes
```

§. @I "C_Interop.hweb" *Section(s) skipped…*

§. @I "Macros.hweb" *Section(s) skipped…*

§. `@I "Fortran.formatting.hweb"` *Section(s) skipped...*

§. `@I "Fortran.macros.hweb"` *Section(s) skipped...*

§. `@I "Fortran.F2x.hweb"` *Section(s) skipped...*