28

ISO TR 15581 Allocatable Enhancements

Aim

The aim of this chapter is to provide a small number of examples illustrating some of the features introduced with ISO TR 15581:

- Allocatable dummy arrays.
- Allocatable function results.
- Allocatable structure components.

28 ISO TR 15581 Allocatable Enhancements

In this chapter we provide three examples that illustrate the features introduced by TR 15581. The facilities mean that we do not have to use pointers and this has several efficiency benefits as the compiler does not have to worry about aliasing and whether it can deallocate temporaries or not. There is also the issue of contiguous memory allocation for allocatable arrays, which can't be guaranteed when using pointers and sections and strides other than unity.

28.1 Allocatable dummy array example

In the Quicksort example the actual array allocation took place in the main program. In this example we do the allocation in the Read_Data subroutine:

```
PROGRAM ch2801
IMPLICIT NONE
INTEGER :: How_Many
CHARACTER (LEN=20) :: File_Name
REAL , ALLOCATABLE , DIMENSION(:) :: Raw_Data
integer , dimension(8) :: timing

INTERFACE
   SUBROUTINE Read_Data(File_Name,Raw_Data,How_Many)
      IMPLICIT NONE
      CHARACTER (LEN=*) , INTENT(IN) :: File_Name
      INTEGER , INTENT(IN) :: How_Many
      REAL , INTENT(OUT) , ALLOCATABLE , &
      DIMENSION(:) :: Raw_Data
   END SUBROUTINE Read_Data
END INTERFACE

INTERFACE
   SUBROUTINE Sort_Data(Raw_Data,How_Many)
      IMPLICIT NONE
      INTEGER , INTENT(IN) :: How_Many
      REAL , INTENT(INOUT) , DIMENSION(:) :: Raw_Data
   END SUBROUTINE Sort_Data
END INTERFACE

INTERFACE
   SUBROUTINE Print_Data(Raw_Data,How_Many)
      IMPLICIT NONE
      INTEGER , INTENT(IN) :: How_Many
```

```
    REAL , INTENT(IN) , DIMENSION(:) :: Raw_Data
  END SUBROUTINE Print_Data
END INTERFACE
  PRINT * , ' How many data items are there?'
  READ  * , How_Many
  PRINT * , ' What is the file name?'
  READ '(A)',File_Name
  call date_and_time(values=timing)
  print * , ' initial'
  print * , timing(6),timing(7),timing(8)
  CALL Read_Data(File_Name,Raw_Data,How_Many)
  call date_and_time(values=timing)
  print * , ' read and allocate'
  print * , timing(6),timing(7),timing(8)
  CALL Sort_Data(Raw_Data,How_Many)
  call date_and_time(values=timing)
  print * , ' sort'
  print * , timing(6),timing(7),timing(8)
  CALL Print_Data(Raw_Data,How_Many)
  call date_and_time(values=timing)
  print * , ' print'
  print * , timing(6),timing(7),timing(8)
  PRINT * , ' '
  PRINT *, ' Data written to file SORTED.DAT'

END PROGRAM ch2801

SUBROUTINE Read_Data(File_Name,Raw_Data,How_Many)
IMPLICIT NONE
CHARACTER (LEN=*) , INTENT(IN) :: File_Name
INTEGER , INTENT(IN) :: How_Many
REAL , INTENT(OUT) , ALLOCATABLE , &
  DIMENSION(:) :: Raw_Data

INTEGER :: I
  ALLOCATE(Raw_Data(1:How_Many))
  OPEN(FILE=File_Name,UNIT=1)
  DO I=1,How_Many
    READ (UNIT=1,FMT=*) Raw_Data(I)
  ENDDO

END SUBROUTINE Read_Data
```

```
SUBROUTINE Sort_Data(Raw_Data,How_Many)
IMPLICIT NONE
INTEGER , INTENT(IN) :: How_Many
REAL , INTENT(INOUT) , DIMENSION(:) :: Raw_Data

  CALL QuickSort(1,How_Many)

CONTAINS

  RECURSIVE SUBROUTINE QuickSort(L,R)
  IMPLICIT NONE
  INTEGER , INTENT(IN) :: L,R
  INTEGER :: I,J,tt
  REAL :: V,T

  i=l
  j=r
  v=raw_data( int((l+r)/2) )
  do
    do while (raw_data(i) < v )
        i=i+1
    enddo
    do while (v < raw_data(j) )
        j=j-1
    enddo
    if (i<=j) then
      t=raw_data(i)
      raw_data(i)=raw_data(j)
      raw_data(j)=t
      i=i+1
      j=j-1
    endif
    if (i>j) exit
  enddo

  if (l<j) then
    call quicksort(l,j)
  endif

  if (i<r) then
    call quicksort(i,r)
```

```
  endif

  END SUBROUTINE QuickSort

END SUBROUTINE Sort_Data

SUBROUTINE Print_Data(Raw_Data,How_Many)
IMPLICIT NONE
INTEGER , INTENT(IN) :: How_Many
REAL , INTENT(IN) , DIMENSION(:) :: Raw_Data
INTEGER :: I
  OPEN(FILE='SORTED.DAT',UNIT=2)
  DO I=1,How_Many
    WRITE(UNIT=2,FMT=*) Raw_Data(I)
  ENDDO
  CLOSE(2)
END SUBROUTINE Print_Data
```

We now have a choice of where we do the allocation. This is thus more flexible than having to do all allocation in the main program, which is effectively a more Fortran 77 style of programming.

28.2 Allocatable function result example

A function may return an array, and in this example the array allocation takes place in the function:

```
PROGRAM ch2802
IMPLICIT NONE

INTERFACE
  FUNCTION Running_Average(R,How_Many) RESULT(Rarray)
    IMPLICIT NONE
    INTEGER , INTENT(IN) :: How_Many
    REAL , ALLOCATABLE , DIMENSION (:) , &
      INTENT(IN) :: R
    REAL , ALLOCATABLE , DIMENSION(:) :: Rarray
  END FUNCTION Running_Average
END INTERFACE

INTERFACE
  SUBROUTINE Read_Data(File_Name,Raw_Data,How_Many)
    IMPLICIT NONE
```

```
    CHARACTER (LEN=*) , INTENT(IN) :: File_Name
    INTEGER , INTENT(IN) :: How_Many
    REAL , INTENT(OUT) , ALLOCATABLE , &
     DIMENSION(:) :: Raw_Data
  END SUBROUTINE Read_Data
END INTERFACE

INTEGER :: How_Many
CHARACTER (LEN=20) :: File_Name
REAL , ALLOCATABLE , DIMENSION(:)   :: Raw_Data
REAL , ALLOCATABLE , DIMENSION(:)   :: RA
INTEGER :: I
  PRINT * , ' How many data items are there?'
  READ   * , How_Many
  PRINT * , ' What is the file name?'
  READ '(A)',File_Name
  CALL Read_Data(File_Name,Raw_Data,How_Many)
  ALLOCATE(RA(1:How_Many))
  RA=Running_Average(Raw_Data,How_Many)
  DO I=1,How_Many
    PRINT *,Raw_Data(I),'    ' ,RA(I)
  END DO
END PROGRAM ch2802

FUNCTION Running_Average(R,How_Many) RESULT(Rarray)
INTEGER , INTENT(IN) :: How_Many
REAL , INTENT(IN) , ALLOCATABLE , DIMENSION(:) :: R
REAL , ALLOCATABLE , DIMENSION(:) :: Rarray
INTEGER :: I
REAL :: Sum=0.0
  ALLOCATE(Rarray(1:How_Many))
  DO I=1,How_Many
    Sum = Sum + R(I)
    Rarray(I)=Sum/I
  END DO
END FUNCTION Running_Average

SUBROUTINE Read_Data(File_Name,Raw_Data,How_Many)
IMPLICIT NONE
CHARACTER (LEN=*) , INTENT(IN) :: File_Name
INTEGER , INTENT(IN) :: How_Many
REAL , INTENT(OUT) , ALLOCATABLE , &
```

```
  DIMENSION(:) :: Raw_Data
INTEGER :: I
  ALLOCATE(Raw_Data(1:How_Many))
  OPEN(FILE=File_Name,UNIT=1)
  DO I=1,How_Many
    READ (UNIT=1,FMT=*) Raw_Data(I)
  ENDDO
END SUBROUTINE Read_Data
```

This is a much more Fortran 90 way of thinking.

28.3 Allocatable structure component example

This example illustrates the use of ragged arrays without the use of pointers:

```
PROGRAM ch2803
IMPLICIT NONE
TYPE Ragged
  REAL , DIMENSION(:) , ALLOCATABLE :: Ragged_row
END TYPE Ragged
INTEGER :: I
INTEGER , PARAMETER :: N=3
TYPE (Ragged) , DIMENSION(1:N) :: Lower_Diag
  DO I=1,N
    ALLOCATE(Lower_Diag(I)%Ragged_Row(1:I))
    PRINT *,' Type in the values for row ' , I
    READ *,Lower_Diag(I)%Ragged_Row(1:I)
  END DO
  DO I=1,N
    PRINT *,Lower_Diag(I)%Ragged_Row(1:I)
  END DO
END PROGRAM ch2803
```

28.4 Summary

These features provide us with a safer way of addressing certain types of problems that would previously have had to be tackled using pointers.

28.5 Problem

These features are not available in all compilers. Try each example out with your compiler to determine the degree of standard conformance.