TECHNICAL REPORT

ISO/IEC TR 15581

First edition
1998-12-01

Information technology — Programming languages — Fortran — Enhanced data type facilities

Technologies de l'information — Langages de programmation — Fortran — Facilités de type de données améliorées

Reference number
ISO/IEC TR 15581:1998(E)

Contents

		Page
1	General	**1**
	1.1 Scope	**1**
	1.2 Normative reference	**1**
2	Requirements	**2**
	2.1 Allocatable Attribute Regularization	**2**
	2.2 Allocatable Arrays as Dummy Arguments	**2**
	2.3 Allocatable Array Function Results	**3**
	2.4 Allocatable Array Components	**4**
3	Required editorial changes to ISO/IEC 1539-1:1997	**7**

ISO/IEC Copyright Office • Case postale 56 • CH-1211 Genève 20 • Switzerland

Printed in Switzerland

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The main task of technical committees is to prepare International Standards, but in exceptional circumstances a technical committee may propose the publication of a Technical Report of one of the following types:

— type 1, when the required support cannot be obtained for the publication of an International Standard, despite repeated efforts;

— type 2, when the subject is still under technical development or where for any other reason there is the future but not immediate possibility of an agreement on an International Standard;

— type 3, when a technical committee has collected data of a different kind from that which is normally published as an International Standard ("state of the art", for example).

Technical Reports of types 1 and 2 are subject to review within three years of publication, to decide whether they can be transformed into International Standards. Technical Reports of type 3 do not necessarily have to be reviewed until the data they provide are considered to be no longer valid or useful.

ISO/IEC TR 15581, which is a Technical Report of type 2, was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*.

This Technical Report specifies an extension of the programming language Fortran, specified by ISO/IEC 1539-1:1997.

It is the intention of ISO/IEC JTC 1/SC 22 that the semantics and syntax described in this Technical Report be incorporated in the next revision of ISO/IEC 1539-1:1997 exactly as they are specified here unless experience in the implementation and use of this feature has identified any errors which need to be corrected, or changes are required in order to achieve proper integration, in which case every reasonable effort will be made to minimize the impact of such integration changes on existing commercial implementations.

Introduction

There are many situations when programming in Fortran where it is necessary to allocate and deallocate arrays of variable size but the full power of pointer arrays is unnecessary and undesirable. In such situations the abilities of a pointer array to alias other arrays and to have non-unit (and variable at execution time) strides are unnecessary, and they are undesirable because this limits optimization, increases the complexity of the program, and increases the likelihood of memory leakage. The ALLOCATABLE attribute solves this problem but can currently only be used for locally stored arrays, a very significant limitation. The most pressing need is for this to be extended to array components; without allocatable array components it is overwhelmingly difficult to create opaque data types with a size that varies at runtime without serious performance penalties and memory leaks.

A major reason for extending the ALLOCATABLE attribute to include dummy arguments and function results is to avoid introducing further irregularities into the language. Furthermore, allocatable dummy arguments improve the ability to hide inessential details during problem decomposition by allowing the allocation and deallocation to occur in called subprograms, which is often the most natural position. Allocatable function results ease the task of creating array functions whose shape is not determined initially on function entry, without negatively impacting performance.

This extension is being defined by means of a Technical Report in the first instance to allow early publication of the proposed definition. This is to encourage early implementation of important extended functionalities in a consistent manner and will allow extensive testing of the design of the extended functionality prior to its incorporation into the language by way of the revision of ISO/IEC 1539-1.

Information technology - Programming languages - Fortran - Enhanced data type facilities

1 General

1.1 Scope

This Technical Report specifies an extension to the data-type facilities of the programming language Fortran. The current Fortran language is specified by ISO/IEC 1539-1 : 1997. The proposed extension allows dummy arguments, function results, and components of derived types to be allocatable arrays.

Clause 2 of this technical report contains a general informal but precise description of the proposed extended functionalities. Clause 3 contains detailed editorial changes which if applied to the current International Standard would implement the revised language specification.

1.2 Normative reference

The following normative document contains provisions which, through reference in this text, constitute provisions of this Technical Report. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this Technical Report are encouraged to investigate the possibility of applying the most recent edition of the normative document indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards.

ISO/IEC 1539-1 : 1997 *Information technology* - *Programming languages* - *Fortran* - *Part 1: Base language.*

2 Requirements

The following subclauses contain a general description of the extensions required to the syntax and semantics of the current Fortran language to provide facilities for regularization of the ALLOCATABLE attribute.

2.1 Allocatable Attribute Regularization

In order to avoid irregularities in the language, the ALLOCATABLE attribute needs to be allowed for all data entities for which it makes sense. Thus, this attribute which was previously limited to locally stored array variables is now allowed on

- array components of structures,
- dummy arrays, and
- array function results.

Allocatable entities remain forbidden from occurring in all places where they may be storage-associated (COMMON blocks and EQUIVALENCE statements). Allocatable array components may appear in SEQUENCE types, but objects of such types are then prohibited from COMMON and EQUIVALENCE.

The semantics for the allocation status of an allocatable entity remain unchanged:

- If it is in a main program or has the SAVE attribute, it has an initial allocation status of not currently allocated. Its allocation status changes only as a result of ALLOCATE and DEALLOCATE statements.
- If it is a module variable without the SAVE attribute, the initial allocation status is not currently allocated and the allocation status may become not currently allocated (by automatic deallocation) whenever execution of a RETURN or END statement results in no active procedure having access to the module.
- If it is a local variable (not accessed by use or host association) and does not have the SAVE attribute, the allocation status becomes not currently allocated on entry to the procedure. On exit from this procedure, if it is currently allocated it is automatically deallocated and the allocation status changes to not currently allocated.

Since an allocatable entity cannot be an alias for an array section (unlike pointer arrays), it may always be stored contiguously.

2.2 Allocatable Arrays as Dummy Arguments

An allocatable dummy argument array shall have associated with it an actual argument which is also an allocatable array.

On procedure entry the allocation status of an allocatable dummy array becomes that of the associated actual argument. If the dummy argument is not INTENT(OUT) and the actual argument is currently allocated, the value of the dummy argument is that of the associated actual argument.

While the procedure is active, an allocatable dummy argument array that does not have INTENT(IN) may be allocated, deallocated, defined, or become undefined. Once any of these events have occurred no reference to the associated actual argument via another alias is permitted .

On exit from the routine the actual argument has the allocation status of the allocatable dummy argument (there is no change, of course, if the allocatable dummy argument has INTENT(IN)). The usual rules apply for propagation of the value from the dummy argument to the actual argument.

No automatic deallocation of the allocatable dummy argument occurs as a result of execution of a RETURN or END statement in the procedure of which it is a dummy argument.

Note that an INTENT(IN) allocatable dummy argument array cannot have its allocation status altered within the called procedure. Thus the main difference between such a dummy argument and a normal dummy array is that it might be unallocated on entry (and throughout execution of the procedure).

Example

```
SUBROUTINE LOAD(ARRAY, FILE)
    REAL, ALLOCATABLE, INTENT(OUT) :: ARRAY(:, :, :)
    CHARACTER(LEN=*), INTENT(IN) :: FILE
    INTEGER UNIT, N1, N2, N3
    INTEGER, EXTERNAL :: GET_LUN
    UNIT = GET_LUN()      ! Returns an unused unit number
    OPEN(UNIT, FILE=FILE, FORM='UNFORMATTED')
    READ(UNIT) N1, N2, N3
    IF (ALLOCATED(ARRAY)) DEALLOCATE(ARRAY)
    ALLOCATE(ARRAY(N1, N2, N3))
    READ(UNIT) ARRAY
    CLOSE(UNIT)
END SUBROUTINE LOAD
```


2.3 Allocatable Array Function Results

An allocatable array function shall have an explicit interface.

On entry to an allocatable array function, the allocation status of the result variable becomes not currently allocated.

The function result variable may be allocated and deallocated any number of times during the execution of the function; however, it shall be currently allocated and have a defined value on exit from the function. Automatic deallocation of the result variable does not occur immediately on exit from the function, but after execution of the statement in which the function reference occurs.[1]

[1] This storage can thus be reclaimed at the same time as that of array temporaries and the results of *explicit-shape-spec* functions referenced in the expression.

Example

```
FUNCTION INQUIRE_FILES_OPEN() RESULT(OPENED_STATUS)
  LOGICAL,ALLOCATABLE :: OPENED_STATUS(:)
  INTEGER I,J
  LOGICAL TEST
  DO I=1000,0,-1
    INQUIRE(UNIT=I,OPENED=TEST,ERR=100)
    IF (TEST) EXIT
100 CONTINUE
  END DO
  ALLOCATE(OPENED_STATUS(0:I))
  DO J=0,I
    INQUIRE(UNIT=J,OPENED=OPENED_STATUS(J))
  END DO
END FUNCTION INQUIRE_FILES_OPEN
```

2.4 Allocatable Array Components

Allocatable array components are defined to be **ultimate components** just as pointer components are, because the value (if any) is stored separately from the rest of the structure and this storage does not exist (because the array is unallocated) when the structure is created. As with ultimate pointer components, variables containing ultimate allocatable array components are forbidden from appearing directly in input/output lists - the user shall list any allocatable array or pointer component for i/o.

As per allocatable arrays currently, they are forbidden from storage association contexts (so any variable containing an ultimate allocatable array component cannot appear in COMMON or EQUIVALENCE); this maintains the clarity and optimizability of allocatable arrays. However, allocatable array components are permitted in SEQUENCE types, which allows the same type to be defined separately in more than one scoping unit.

Deallocation of a variable containing an ultimate allocatable array component automatically deallocates all such components of the variable that are currently allocated.

In a structure constructor for a derived type containing an allocatable array component, the expression corresponding to the allocatable array component must be one of the following:

- an argumentless reference to the intrinsic function `NULL();`the allocatable array component receives the allocation status of not currently allocated.
- a variable that is itself an allocatable array; the allocatable array component receives the allocation status of the variable, and, if allocated, the shape and value of the variable.
- any other array expression; the allocatable array component receives the allocation status of currently allocated with the same shape and value as the expression.

For intrinsic assignment of objects of a derived type containing an allocatable array component, the allocatable array component of the variable on the left-hand-side receives the allocation status and, if allocated, the shape and value of the corresponding component of the expression. This occurs as if the following sequence of steps is carried out:[2]

1. If the component of the variable is currently allocated, it is deallocated.
2. If the corresponding component of the expression is currently allocated, the component of the variable is allocated with the same shape. The value of the component of the expression is then assigned to the corresponding component of the variable using intrinsic assignment.

Note that this definition of assignment facilitates certain optimizations when the allocatable array component of the expression is allocated. In particular:

1. If the corresponding component of the variable is allocated with the same (or larger) size, its storage can be re-used without the overhead of an additional allocation or deallocation;
2. If the expression is a function reference, the processor can simply copy the descriptor instead of the allocatable array contents and omit the deallocation of this component.

[2] This ensures that any pointers that point to the previous contents of the allocatable array component of the variable become undefined. Implementations are thus free to skip the allocation-deallocation (or not) when the component of the variable happens to be allocated with the same shape as the corresponding component of the expression, whichever is most efficient.