Subject: Issue 280
From: Van Snyder
References: 00-195r2

1 Introduction

Issue 280 says "ALLOCATE (... SOURCE = ...) begs for an example. Here's one.

2 Edits

Edits refer to 00-007r2. Page and line numbers are displayed in the margin. Absent other instructions, a page and line number or line number range implies all of the indicated text is to be replaced by immediately following text, while a page and line number followed by + (-) indicates that immediately following text is to be inserted after (before) the indicated line. Remarks for the editor are noted in the margin, or appear between [and] in the text.

[Editor: Replace issue 280:] 103:36-38

NOTE 6.17$\frac{1}{2}$

An example of an ALLOCATE statement in which the value and dynamic type are determined by reference to another object is:

```
CLASS(*), ALLOCATABLE :: NEW
CLASS(*), POINTER :: OLD
! ...
ALLOCATE ( NEW, SOURCE = OLD ) ! Allocate NEW with the value and
                               ! dynamic type of OLD
```

A more extensive example is given in C.3.1$\frac{1}{2}$.

C.3.1$\frac{1}{2}$ Allocation with dynamic type (6.3.1 422:8+

The following example illustrates the use of allocation with the value and dynamic type of the allocated object given by another, to copy a list of objects of any extensible type. It copies the list starting at `IN_LIST`. After copying, each element of the list starting at `LIST_COPY` has a polymorphic component, `ITEM`, for which both the value and type are taken from the `ITEM` component of the corresponding element of the list starting at `IN_LIST`.

NOTE 6.17$\frac{1}{2}$

```
TYPE :: LIST ! A list of anything of extensible type
  TYPE(LIST), POINTER :: NEXT => NULL()
  CLASS(*), ALLOCATABLE :: ITEM
END TYPE LIST

...
TYPE(LIST), POINTER :: IN_LIST, LIST_COPY => NULL()
TYPE(LIST), POINTER :: IN_WALK, NEW_TAIL
! Copy IN_LIST to LIST_COPY
IF ( ASSOCIATED(IN_LIST) ) THEN
  IN_WALK => IN_LIST
  ALLOCATE ( LIST_COPY )
  NEW_TAIL => LIST_COPY
  DO
    ALLOCATE ( NEW_TAIL % ITEM, SOURCE = IN_WALK % ITEM )
    IN_WALK => IN_WALK % NEXT
    IF ( .NOT. ASSOCIATED(IN_WALK) ) EXIT
    ALLOCATE ( NEW_TAIL % NEXT )
    NEW_TAIL => NEW_TAIL % NEXT )
  END DO
END IF
```