AFNOR proposal on object orientation in Fortran 2000
Type extension

Extend an existing derived type by adding zero or more additional components

```
TYPE,EXTENDS() :: POINT_2D
  REAL :: X,Y
END TYPE POINT_2D

TYPE,EXTENDS(POINT_2D) :: POINT_3D
  REAL :: Z
END TYPE POINT_3D

TYPE,EXTENDS(POINT_3D) :: POINT_COLOR
  INTEGER :: COLOR
END TYPE POINT_COLOR

TYPE,EXTENDS(POINT_2D) :: POINT_RADIO
  REAL :: FREQUENCY
END TYPE POINT_RADIO
```
Extension type hierarchy

TYPE()

TYPE(POINT_2D)

TYPE(POINT_3D)  TYPE(POINT_RADIO)

TYPE(POINT_COLOR)

Supertype

Subtype or extension type
Supertype cast operation (1)

Consider a subroutine operating on POINT_2D types:

```fortran
SUBROUTINE POLAR(POINT)
  TYPE(POINT_2D), INTENT(IN) :: POINT
  PRINT *, 'polar angle=', ATAN(POINT%Y/POINT%X)
  PRINT *, 'modulus=', SQRT(POINT%X**2 + POINT%Y**2)
END SUBROUTINE POLAR
```

Consider now using this subroutine on a variable of type POINT_COLOR:

```fortran
  TYPE(POINT_COLOR) :: A

  CALL POLAR(A) ! Compile-time error
  CALL POLAR(A%POINT_3D%POINT_2D) ! Legal, X3J3 syntax
  CALL POLAR(POINT_2D%A) ! Legal, AFNOR supertype cast
```
Supertype cast operation (2)

Comparison of AFNOR and X3J3 syntax:

```
TYPE(POINT_COLOR) :: A
```

<table>
<thead>
<tr>
<th>Afnor</th>
<th>X3J3</th>
</tr>
</thead>
<tbody>
<tr>
<td>POINT_3D@A</td>
<td>A%POINT_3D</td>
</tr>
<tr>
<td>POINT_2D@A</td>
<td>A%POINT_3D%POINT_2D</td>
</tr>
<tr>
<td>POINT_2D@A%X</td>
<td>A%POINT_3D%POINT_2D%X</td>
</tr>
<tr>
<td>A%X</td>
<td>A%X</td>
</tr>
</tbody>
</table>

Restriction on X3J3 syntax:

«A component or type parameter declared in an extended type shall not have the same name as the parent type.»  page 56, line 23
Polymorphic variable

Ability for a variable declared with the CLASS keyword to assume differing dynamic types during program execution:

```
CLASS(POINT_2D) :: A ! The dynamic type of A is
    ! POINT_2D, POINT_3D, POINT_RADIO
    ! or POINT_COLOR
CLASS() :: B ! The dynamic type of B is any
    ! extensible type
CLASS(POINT_3D) :: C ! The dynamic type of C is
    ! POINT_3D or POINT_COLOR
```

A polymorphic variable gets its dynamic type via argument association, pointer assignment, NULLIFY, or execution of ALLOCATE or DEALLOCATE statement.
Allocation of a polymorphic variable

Consider a polymorphic variable $A$:

```cpp
CLASS (POINT_2D) :: A
```

**Default allocation:**

```cpp
ALLOCATE(A) ! The dynamic type of $A$ is POINT_2D
```

**Casted allocation, X3J3 syntax:**

```cpp
ALLOCATE(TYPE(COLOR_POINT) :: A) ! The dynamic type of
    A is COLOR_POINT
```

**Casted allocation, AFNOR syntax:**

```cpp
ALLOCATE(COLOR_POINT@A) ! The dynamic type of $A$ is
    COLOR_POINT

ALLOCATE(A, CAST=B) ! The dynamic type of $A$ is the same
    as the dynamic type of $B$
```

The second form of AFNOR casted allocation is not possible with the X3J3 syntax
Dynamic Dispatch

To be able to make a procedure reference where the specific procedure that is called depends on the dynamic type of a polymorphic variable.

```plaintext
CLASS (POINT_2D), POINTER :: A
TYPE (POINT_3D) :: B
A => B  ! The dynamic type of A is POINT_3D
CALL METHOD (A, other parameters)  ! Dynamic dispatch
```

Here, a run-time analysis of this request is made:

- If `CALL METHOD` can operate on objects of type `POINT_3D` (the dynamic type of `A`), the run-time system replace the call with
  ```plaintext
  CALL METHOD (POINT_3D@A, other parameters)
  ```

- If `CALL METHOD` can operate on objects of type `POINT_2D` (the parent type of the dynamic type of `A`), the run-time system replace the call with
  ```plaintext
  CALL METHOD (POINT_2D@A, other parameters)
  ```

- Else, **run-time error**
"Only components of the declared type of a polymorphic object may be designated by component selection". Page 77, line 2

CLASS (POINT_2D) :: A
TYPE (POINT_COLOR) :: B
A => B ! The dynamic type of A is B
X = A%COLOR ! Compile-time error
X = POINT_COLOR@A%COLOR ! Legal, AFNOR subtype cast

SELECT TYPE(A) ASSOCIATE(point) ! Legal, X3J3 syntax
TYPE IS (POINT_COLOR)
X = point%COLOR
END SELECT ! End of X3J3 syntax

The AFNOR syntax is type-unsafe (a run-time error may occurs); the X3J3 syntax is type-safe (a run-type error cannot occurs).
Unresolved issues with type cast

- Should type cast be available in Fortran2000
  - X3J3: Supertype cast is type safe, but it can be replaced with \%-operations with restrictions; subtype cast is type unsafe and should not be available
  - AFNOR: Both supertype cast and subtype cast operations should be available

- Should casted-allocation be available in Fortran2000
  - Two forms of casted allocation are required.
    - \texttt{ALLOCATE\,(TYPE\,(COLOR\_POINT) \::\: A) or ALLOCATE\,(COLOR\_POINT\_A)}
    - \texttt{ALLOCATE\,(A,\,CAST=B)}
Unresolved issues with dynamic dispatch

- Should dynamic dispatch be available only for type-bound procedures?
  \[ \textbf{AFNOR}: \text{Should be available also to ordinary procedures.} \]

- What happens if a procedure is called with many polymorphic variables:
  
  \[
  \text{CLASS(POINT}_2\text{D)} :: A,B \\
  \text{CALL METHOD(A,B,other parameters)}
  \]
  
  \[ \textbf{AFNOR}: \text{The order of resolution should follow some rules.} \]

- Can dynamic dispatch be made type-safe without loosing flexibility?
  \[ \textbf{AFNOR}: \text{No. Exception handling may be added to deal with the type-unsafe characteristic of dynamic dispatch.} \]
CONSTRUCTOR and DESTRUCTOR capabilities (1)

We propose to supplement the ALLOCATE statement with a «CONSTRUCTOR» capability and to supplement the DEALLOCATE statement with a «DESTRUCTOR» capability.

Defined as a type bound procedure with name CONSTRUCTOR or DESTRUCTOR:

```fortran
TYPE CHAR_OBJ
  INTEGER :: NTABLE = 0
  CHARACTER(LEN=1),POINTER,DIMENSION(:) :: PTEXT
  CONTAINS
    PROCEDURE,PASS_OBJ :: CONSTRUCTOR => SUB00
    PROCEDURE,PASS_OBJ :: DESTRUCTOR => SUB11
END TYPE CHAR_OBJ
```

June, 1999
CONSTRUCTOR and DESTRUCTOR capabilities (2)

The constructor/destrokers are implemented as elemental subroutines:

```fortran
ELEMENTAL SUBROUTINE SUB00 (AA, LENGt)
  TYPE (CHAR_OBJ) :: AA
  AA%NTABLE = LENGt
  ALLOCATE (AA%PNEXT (LENGt))
  AA&PTEXT = ' '
END SUBROUTINE SUB00

ELEMENTAL SUBROUTINE SUB11(AA)
  IF (AA%NTABLE > 0) DEALLOCATE (AA%PNEXT)
END SUBROUTINE SUB11
```

In the calling procedure, we write:

```fortran
TYPE (CHAR_OBJ), DIMENSION (:), ALLOCATABLE :: STRING
ALLOCATE (STRING (10), LENGt=5)
...
DEALLOCATE (STRING)
```