4.9 Polymorphism

Replace section by

Fortran supports object orientation with single inheritance. A derived type `ta` may be extended to form a new type `tb` with all the components of type `ta` plus possible additional components. The extended type `tb` also has a parent component of type `ta` with the name `ta` and the type and type parameters of the parent type. Access to the components is illustrated by the following example.

```
type ta
   real :: x
end type
type, extends (ta) :: tb
   integer :: i
end type
type(tb) :: bobj
. . .
bobj%x  = 1
bobj%ta%x = 2 ! Overwrites the previous assignment of 1
```

A variable can be declared as polymorphic; it has a declared type and a dynamic type that is permitted to be any extension of the declared type. A type declaration can bind existing procedures to the type; each has a binding name that may be the same as the name of the existing procedure. The existing procedure usually has a dummy argument of the type that is given the `pass` attribute. A type-bound procedure is invoked as if it were a component of the object; if the procedure has an argument with the `pass` attribute, the corresponding actual argument is omitted from the argument list and the invoking object is passed automatically. Here is an example

```
module m
   type ta
      real :: x = 7.2
   end type
   type, extends (ta) :: tb
      integer :: i
   contains
      procedure :: proc => foo  ! first argument implictly given
                                ! the pass attribute
   end type
contains
   real function foo( arg )
      class(tb) :: arg
      foo = arg%x
   end function
end module m
. . .
   use m
   type(tb) :: bobj
   real :: y
```

```
    y = bobj%proc()    ! y is assigned the value 7.2
```

Binding names are inherited by extensions of the type but can be overridden by a specification for the same name in the definition of an extended type. Which procedure is invoked in a type-bound reference is determined by the dynamic type of the object through which the procedure is referenced. To execute alternative code depending on the dynamic type of a polymorphic entity and to gain access to the dynamic parts, the `select type` construct is provided.

7 Language specific vulnerabilities for Fortran

Replace by

7.1 Source form

Commented [SM1]: STEVE to write.

7.1.1 Applicability to language

Fortran permits a source form called "fixed" where blanks are not significant in parsing the source code, and a source form called "free" where blanks are significant. A famous example of the vulnerability associated with fixed source form is

```
do 25 i = 1.10
```

Is interpreted as an assignment of 1.1 to the (undeclared) floating point variable `do25i` instead of as the loop header

```
do 25 i = 1,10
```

In addition, fixed source form ignores text beyond line position 72, whereas for free form code, all characters within the legal line length are significant (except beyond the character !). The vulnerability associated with fixed form source code is that any text placed beyond line position 72 is ignored, which can change the semantics.

7.1.2 Avoidance mechanisms for language users

- Avoid fixed source form in all programs.
- Use `implicit none` to require that all variables are declared, see 6.17 Choice of clear names [NAI].