Interface block

Function as a parameter

We have a procedure for solving an equation. It would be nice if we could give the equation as a parameter to the function. Then we could use the same procedure to solve different equations in the same program.

Begin by considering a trivial example. Let the main program be in the file `main.f90`:

```
program main
  real :: x(5) = 1.0, summa
  real, external :: f
  summa=sub(f, x)
  write(*,*) summa

contains

real function sub(f, x)
  real, external :: f
  real x(:)
  sub=f(x)
end function
end program
```

The function `sub` gets the function `f` as a parameter. The procedure appearing as a parameter must be compiled separately. The attribute `external` tells that `f` is a separately compiled external object.

Assume the function is in a file `func.f90`:

```
real function f(x)
real x(:)
f=sum(x)
end function
```

The program is compiled and linked the usual way:

```
f95 -o ohjelma main.f90 func.f90
```

Properties of a separately compiled procedure are not known when compiling the main program. Hence not even the number of parameters and their types can be checked. When the program is executed an error in parameters may lead to the famous message 'segmentaion fault'.

However, the proper form of the call of the procedure can be defined in an interface block:

```
real function integrate(f, a, b)
implicit none

interface
  real function f(x)
  real, intent(in) :: x
  end function
end interface

real, intent(in) :: a, b
sum=f(a)+f(b)
...
integrate=sum
end function
```

Here `interface` contains the declaration of the function and its arguments but no executable code nor local variables. The interface block tells the compiler how the function must be called; thus the compiler can check the correctness of the call.

The interface block can be created easily by copying the first few lines of the function definition.