Computer Seminar 2 (Spring 2025)

Class Notes

C2TB1702

Maximilian Fernaldy

Class 1

- No distinction between single quotes and double quotes. Both specify string literals.
- No distinction between uppercase and lowercase letters (except in string literals).
- Programs cannot be nested. The following is not valid Fortran:

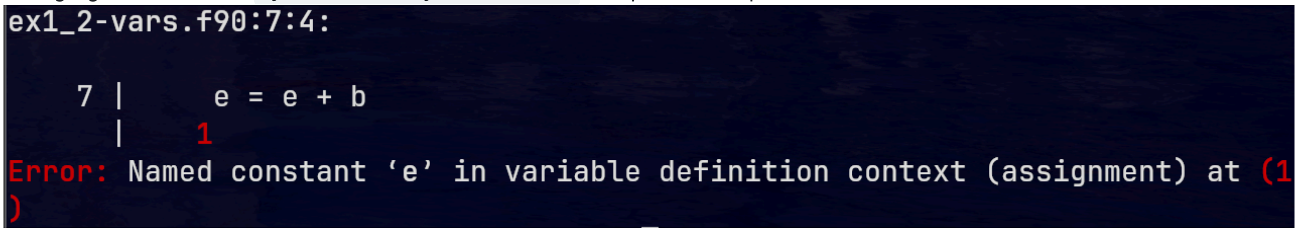
  ```fortran
  program hello
      print *, "Hello, world!"
      program hi
          print *, "Hi, world!"
      end program
  end program
  ```

  which is why it is sufficient to type `end program` instead of `end program hello` at the end of every program.
- To define a variable, write the type of the variable, then a double colon (like the scope resolution operator in C++) and the variable name/identifier.

  ```fortran
  integer :: myInt = 6
  ```

- To define a constant, type `parameter` as an attribute when defining a variable:

  ```fortran
  real,parameter :: myConstant = 8 !8 or 8.0 is fine
  ```

  changing this value like `myConstant = myConstant + 8` will yield a compilation error like so:
  

  ```
  ex1_2-vars.f90:7:4:

      7 |      e = e + b
        |     1
  Error: Named constant ‘e’ in variable definition context (assignment) at (1
  )
  ```
- Real literals are in the format `<mantissa>e<exponent>`. For example,

  ```fortran
  real :: myNumber1 = 9.8d0 !Equal to 9.8
  real :: myNumber2 = 9.8d1 !Equal to 98
  real :: myNumber3 = 9.8d-1 !Equal to 0.98
  ```

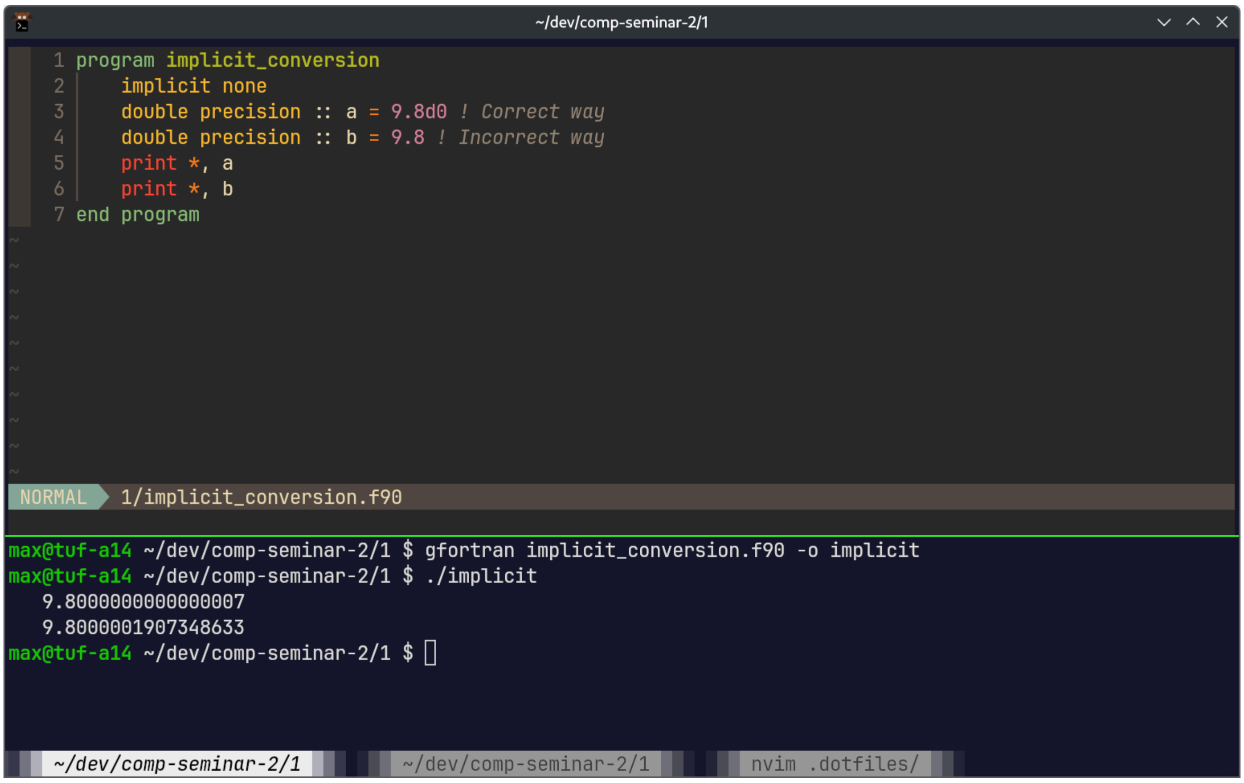
  It is also possible to use a decimal point to indicate a real literal:

  ```fortran
  real :: myNumber4 = 1.0
  real :: myNumber5 = 1. !No trailing zero is also acceptable, just like C!
  ```

- Same is true for double, just with d instead of e: `<mantissa>d<exponent>`.

  ```fortran
  double precision :: myDouble = 9.8d0
  ```

  > ⚠ Warning
  >
  > **Be careful!** you should always use the above format to define a double precision number. Otherwise, the number you use to define the variable (the literal) will be a *real literal* instead of a *double literal*. This triggers an *implicit*

> *conversion*, which means the compiler will convert the data for you without raising any errors or warnings. Converting a real literal into a double literal means you will lose some precision, defeating the point of using a double precision type in the first place:
>
> 
>
> ```
> program implicit_conversion
>     implicit none
>     double precision :: a = 9.8d0 ! Correct way
>     double precision :: b = 9.8 ! Incorrect way
>     print *, a
>     print *, b
> end program
> ```
>
> ```
> NORMAL  1/implicit_conversion.f90
> ```
>
> ```
> max@tuf-a14 ~/dev/comp-seminar-2/1 $ gfortran implicit_conversion.f90 -o implicit
> max@tuf-a14 ~/dev/comp-seminar-2/1 $ ./implicit
>    9.8000000000000007
>    9.8000001907348633
> max@tuf-a14 ~/dev/comp-seminar-2/1 $
> ```
>
>

- It seems Fortran supports function overloading (like C++), or at least for the intrinsic functions built into the language. For example, these are the intrinsic functions used to find the largest number in an array:

  ```
  max0(i0, i1, i2) ! For integer types
  amax1(a0, a1, a2) ! For real types
  dmax1(d0, d1, d2) ! For double types
  max(myNumber0, myNumber1, MyNumber2) ! Automatic
  ```

  `max()` will automatically replace the function call with the correct one depending on the type of array passed into it (similar to how some stdlib functions in C++ may work slightly differently depending on the data type of the parameters passed into the functions).