Fortran Programming Language: Notes

Moritz M. Konarski

August 26, 2020

Contents

Introduction

- knowing how computers work is good to know
- problem solving and analytical skills are very important
- fortran (FORmula TRANslation) was specifically designed for scientific and engineering applications
- complete fortran reference can be found at https://gcc.gnu.org/wiki/GFortran

Basic Programming

- a fortran program starts with `program <name>` and ends with `end program <name>`, name being an appropriate name
- the name has to start with a letter and can then contain letters, numbers and underscores, at most 32 characters
- capitalization does not make a difference
- comments are designated by exclamation points, everything following them is ignored

```
program hello

! some code

end program hello
```

- simple output is done with `write`

```
! Hello World Program
program hello_world

    write(*,*) "Hello, world!"

end program hello_world
```

- after compiling the code using `gfortran hello_world.f95` the program can be executed
- one useful option is to compile with bounds checking `gfortran -fcheck=bounds ...`

Basic Elements

- variables must be declared at the start of the program
- variable names are 32 chars long at most, start with a letter, and can contain letters, characters, and underscores – capitalization does not matter
- data can either be represented as a literal or as a variable
- the data types are
  - integer: whole number, integer division shenanigans
  - real: floating point numbers
  - complex: a complex number $a + ib$ where a and b are reals
  - character: ASCII characters or strings
  - logical: `.true.` and `.false.` booleans
- `implicit none` at the start of the program will turn off implicit typing which is bad practice anyways

Declarations

- variables must all be declared before executable statements in the program

```
<type> :: <list of var names>
! for example
integer :: age, points
real :: area
complex :: root
logical :: is_true
character :: ans_char
```

- type mismatch is bad and the compiler might actually implicitly convert some things, but that is not best practice
- initializations are done using equal sings

```
integer :: age = 21, points = 10
```

- constants are defined using the `parameter` keyword

```
real, parameter :: pi = 3.14159265
```

- comments should contain useful and reference information about the program
- if a line is to be continued on the next one, an ampersand can be used

```
A = 174.5 * year             &
    + count / 100.0
```

- to declare a string, use

```
    character(11) :: msg = "Hello World"
```

- larger variables of int and float can be declared by giving the byte amounts

```
    integer :: num                  ! 32 bit
    integer*8 :: bignum             ! 64 bit
    integer(kind=8) :: biggernum    ! 64 bit

    real :: rnum                    ! 32 bit
    real*8 :: bigrnum               ! 64 bit
    real(kind=8) :: biggerrnum      ! 64 bit
```

Expressions

- e-notation can be used in upper and lower case

```
2.74E5
2.34345e-13
```

- complex literals are written in parentheses

```
(3.2, 4)
(1, 14)
```

- character literals are normal, escaping double quotes is done by doubling them

```
"this is ""wow"" in fortran"
! turns into "this is "wow" in fortran"
```

- assignment is done by the equal sign `var = expr`
- only one variable can be assigned on each sign
- operations:
  - `+` as addition
  - `-` as subtraction
  - `*` as multiplication
  - `/` as division
  - `**` as exponentiation
- order of operations: unary `-`, exponentiation, multiplication & division, addition & subtraction

Intrinsic functions

- `cos(pi)` and other trigonometric functions
- conversion is done with
  - `real(<int>)`
  - `int(<real>)` truncates real
  - `nint(<real>)` rounds real
- `sin`, `cos`, `tan`, `mod(r1, r2)` (r1 divided by r2), `sqrt`
- when ints and reals are being mixed, it's called mixed mode – it is unintuitive and sometimes weird, explicit conversion should be done
-

Simple Input and Output

Write

- write is the simplest form of output

```
write (*,*) "Hello World"
```

- `(*,*)` refers to send to screen in free format
- it can also show the value of declared variables

```
integer :: num1 = 12
write (*,*) num1
```

- if write is supplied no arguments it prints a new line
- multiple item can be chained by commas to be printed

```
integer :: num1 = 12, num2 = 2
write (*,*) "Number 1 = ", num1, "Number 2 = ", num2
```

Print

- the print statement is more restrictive than write, it only sends output to the screen

```
integer :: num1 = 12, num2 = 2
print *,"Number 1 = ", num1, "Number 2 = ", num2
```

Read

- read retrieves information from the user

```
! reads one input
read (*,*) ans1
! reads two inputs
read (*,*) ans1, ans2
```

- if the input values are not of the appropriate type, an error occurs

Program Development

- program development has the following steps
  1. understand the problem
  2. create the algorithm
  3. implement the program
  4. test and debug the program

Selection Statements

- relational operators in fortran

Operation	normal operator	alternate operator
greater	`>`	`.gt.`
greater or equal	`>=`	`.ge.`
less	`<`	`.lt.`
less or equal	`<=`	`.le.`
equal	`==`	`.eq.`
not equal	`/=`	`.ne.`

- conditional expressions are enclosed in brackets

```fortran
(weight > 0)
```

- logical operators in fortran

```fortran
.and.        ! and statement
.or.         ! or statement
.not.        ! not statement
```

- with these operators logical statements can be combined

```fortran
( (weight > 0) .and. (weight < 120) )
```

- if then statement

```fortran
if ( <condition> ) then
    <do stuff>
end if
```

- simple if then statement

```fortran
if ( <condition> ) <single statement>
```

- if then else statement

```fortran
if ( <condition> ) then
    <stuff>
else
    <stuff>
end if
```

- else if statement

```
if ( <condition> ) then
    <stuff>
else if ( <condition> ) then
    <stuff>
else
    <stuff>
end if
```

- select case statement

```
select case ( var )
    case ( selector )
        ...
    case default
        ...
end select
```

- selectors can be

```
! only 1 value
( value )
! from val1 to val2 inclusive
( val1 : val2 )
! greater or equal val1
( val1 : )
! less or equal val2
( : val2 )
```

- there can also be lists of numbers in the selector

```
case (1,3,4,5,6,7)
    ...
```

Looping

Counter Controlled Looping

```
do count_var = start, stop, step
    ...
end do
```

- all values in the loop are integers
- if the step value is omitted, it is assumed 1
- the three values are not recomputed during looping
- if `count` is less than or equal to `stop`, the loop continues
- the `break` statement in fortran is `exit`
- `continue` in fortran is `cycle`

Conditional Controlled Looping

```
do while ( condition )
    ...
end do
```

- another form contains an exit statement somewhere in the loop

```
do
    ...
    if ( condition ) exit
    ...
end do
```

- conditional loops can be used to loop with reals, not supported by counter loops

Formatted Input/Output

- the second `*` in read and write statements can be replaced by format instructions

```
read  (*, '( formatting )') <vars>
write (*, '( formatting )') <vars / expressions>
```

- convention of symbols to be used
  - `w` – number of used positions
  - `m` – minimum number of positions
  - `d` – digits to the right of the decimal point
  - `n` – number or count
  - `c` – column number
  - `r` – repeat count
- basic table of common formats

Description	Specifier
Integers	`rIw` or `rIw.m`
Reals	`rFw.d`
Logicals	`rlw`
Characters	`rA` or `rAw`
Hor. pos. space	`nX`
Hor. pos. column	`Tc`
Vertical Space	`n/`

Integer formatting

- use `i` instead of `I` and otherwise it's ok

```
write (*,'(i5.6)') 1234
! > 001234
```

Real formatting

- `w` here is the total number of places, including the decimal point, `f` instead of `F`

```
write (*,'(f9.4)') 234.34556
! > " 234.3455"
```

Horizontal Positioning

- `nX` inserts `n` spaces, `Tc` goes to column `c`

Logical Formatting

- `L` is `l` here, the variables are printed as `T` or `F`
- we can specify how many places should be displayed

Character Formatting

- if no length is given, the full length is used
- if the given length is too short, the string is cut off
- the `trim()` function remove excess white space

Advance Clause

- if one does not want a new line after `write`

```
write(*, '(a)', advance=no) "Enter number: "
```

- options are `yes` and `no`