

Fortran 77 Tutorial

Hsiu-Pin Chen

January 4th, 2007



Concepts

- **Program:** to provide the computer a set of working instructions and make it perform certain tasks in a desired way.
- **Programming language:** Human-readable language that can be translated into machine-readable language through its own inherent compiler. Such as C, C++, Fortran.



Programming Language

- **They have fixed rules such as syntax and formats.**
- **These rules have to be followed exactly or the program will generate undesired results.**
- **They are necessary for communication between human and computer.**



Interface between human and computers

- **Human can understand programming language, while computers can only understand machine codes. They are not cross-readable.**
- **Compiler does the translation job between two languages.**



Column position rules

- **Col. 7 - 72 : codes.**
- **Col. 1 : comment symbol**
- **Col. 1 - 5 : statement label**
- **Col. 6 : continuation**



Case Insensitivity

- Fortran is **case insensitive**.

"INTEGER" & "integer" won't be distinguished by the compiler.



Program structure

Program *test*

Declarations of variable

Input statements

Calculations

Output statements

End



Operators & intrinsic functions

- **Mathematical operators:** +, -, *, /, (), **

- **Logical operators:**

 - .GT., .GE., .EQ., .NE., .LT., .LE., .AND.,
.OR., .NOT. (Results would be logical values:
.TRUE. or .FALSE.)

- **Intrinsic functions:**

 - abs, min, max, sqrt, sin, cos, tan, atan, exp,
log, mod(modulus, usage: mod(a, b))

*Pay attention to the **priority level** of these operators*



Arrays

- A set of data that shares the same name and type, and are stored in the memory consecutively.

Data_type_name array_name(size)

Examples:

integer weekdays(7), events(12, 31)

one-dimensional .vs. multi-dimensional array

- Preset values in arrays: block data statement

Examples:

real a,b(5,6),c(12)

data a/0.5/, b/30*0.0/, c/5*2.5, 7*3.0/



Variable declaration and expression

- Calculate $(28 * 57 + 23 * 14) ^ 2 + (28 * 57 + 23 * 14) = ?$

integer element, result

element = $28 * 57 + 23 * 14$

result = element * element + element

- Data_type_name variable_name
variable_name = value



Variable declaration and expression

Type	Example
Integer	10
Real	10.1
Double precision	10.1D0
Logical	.true. Or .false.
Character	'test', '100'

Variables may or may not have initial values, and they can be changed, i.e., be assigned to other values during the program.



Do loops

- Calculate $1*1 + 2*2 + 3*3 + \dots + 100*100 = ?$

program test

```
integer sum,current_square i
sum = 0
```

```
  do i = 1, 100
```

```
    current_square = i*i
```

```
    sum = sum + current_square
```

```
  enddo
```

```
  print *, sum
```

```
end
```



If statements

- What if I don't want the sum to include those terms of $10 * n$, i.e., 10, 20, 30..., 100?

```
do i = 1, 100
  if (mod(i, 10) .NE. 0) then
    current_square = i * i
    sum = sum + current_square
  endif
enddo
```



Subprograms

```
1)  subroutine increment(ans, i)
      integer i, ans
      ans = ans + i*i
end
```

```
program test
  integer sum, i
  sum = 0
  do i = 1, 100
    call increment(sum, i)
  enddo
  print *, sum
end
```



Subprograms

•2) **Integer function increment(i)**

Integer i

Increment = i * i

return

end

program test

integer sum, i

sum = 0

do i = 1, 1000

sum = sum + increment(i)

enddo

print *, sum

end



Methods to pass data between subprograms

- 1) As **arguments (parameters)** in function (subroutine) calls, in which case they are local variables:

call increment(sum, i)
sum = sum + increment(i)

(subroutine)
(function)



Methods to pass data between subprograms

2). As global variables (common blocks) that can be accessed by any subprogram:

```
subroutine increment(i)
  integer i, sum
  common /data/ sum
  sum = sum + i * i
end
```



I/O format

- `read (*, *)` variables
- `write (*, *)` variables
 - The first “ * ” in the brackets means we will read from / write to standard devices like keyboard/ screen.
 - Second “ * ” means the default reading/ writing formats.



I/O format

```
open (15, FILE='input', FORM='FORMATTED'  
,STATUS='UNKNOWN')
```

```
open (16, FILE='output', FORM='FORMATTED'  
,STATUS='UNKNOWN')
```

```
read (15, *) numerator, denominator
```

```
write (16, *) ratio
```

```
close (15)
```

```
close (16)
```

