Programming for Engineers
Fortran: Getting Started

Abu Hasan Abdullah

January 8, 2009

Preview

- How a program is organized
- The different types of data, constants and variables
- Assignment statements for calculating and storing data
- Simple input and output

Course Text:

Mayo W. E. and Cwiakala M. (1995): *Programming with Fortran 77*, ISBN 0-07-041155-7, McGraw-Hill

Our First Fortran Program

- A program is constructed with a text editor. Run your text editor and edit the following Fortran source code into a file named `salam.f`

```
12345678901234567890...
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣...
1 ␣␣␣␣␣␣PROGRAM␣salam
2 ␣␣␣␣␣␣PRINT␣*,␣'Salam,␣world!'
3 ␣␣␣␣␣␣END
```

- As shown above, a program may (optionally) start with the statement

  `PROGRAM` *name*

  and must end with the statement

  `END`

Program Organization

- A program line must follow very specific rules, e.g.
  - Columns 1 thru' 5 reserved for *statement labels*.
  - Column 6 reserved for a *continuation character*.
  - Columns 7 thru' 72 store *Fortran commands*.
  - Columns 73 and higher are ignored. Can be used for *comments*.

- Thus, without the imaginary spaces, the code is

```
1       PROGRAM salam
2       PRINT *, 'Salam, world!'
3       END
```

- Comments, used for documenting various parts of a program, are indicated by a C in column 1 or character ! in any column

Running g77 Fortran Compiler

- Compile salam.f source file by typing

```
[userid@siswa]$ g77 -o salam salam.f
```

  at the console command prompt. If all goes well, object file salam.o is created and linked to the system libraries to produce executable file salam.

- To execute or run a successfully compiled program, type

```
[userid@siswa]$ ./salam
```

- Explore other compiler options by typing

```
[userid@siswa]$ man g77
```

Data Types

- Fortran contains six *intrinsic data types*, built automatically into the language and divided into two categories:

  1. numerical data types
     (a) integer
     (b) real
     (c) double precision
     (d) complex
  2. non-numerical data types
     (a) character
     (b) logical

Data Types
Integer Constants

- Those that represent *whole numbers*.
- Range of values that can be represented on a computer varies from one computer to another.
- A typical range -2^{32-1} to $+2^{32-1}$.
  Approximately $\pm 2 \times 10^9$ for a 32-bit computer.

Data Types
Real Constants

- Represent fractional numbers which maybe positive or negative and always have a decimal point.

- Stored in computer as two components:
  - mantissa—ranging between 0.1 and 1.0.
  - exponent—indicates appropriate power of 10.

  For example 10.2345 is stored with 0.102345 as mantissa, 2 as exponent (i.e. $10.2345 \equiv 0.102345 \times 10^2$).

Data Types
Real Constants

- Used with *scientific notation* to represent a very large or very small numbers. Written as:

  $< \text{mantissa} > \times 10^{<\text{exponent}>}$

  For example, 0.123456×10^5 represents $123,456$.

- Accuracy limited to seven digits, magnitude ranging 10^{-39} to 10^{+38}.

- Samples in **Example 2.4**, **Example 2.5**.

EXAMPLE 2.4

The following examples illustrate valid and invalid uses of real constants:

Valid Examples	Invalid Examples	Comment
-21.4		*Negative required*
+132.7		*Plus sign optional*
0.0000034		*Small numbers permitted*
123 456.0		*Spaces ignored*
	$ 1.23	*Only numbers permitted (no $)*
	0	*Requires a decimal point, otherwise this is an integer*
	123,456.00	*No commas*

EXAMPLE 2.5

The following examples show the use of real constants using scientific notation:

Valid Examples	Invalid Examples	Comment
0.6023E24		*Avogadro's number* 6.023×10^{23}
−0.123E24		*Negative mantissa permitted*
0.123E-24		*Negative exponent permitted*
0.0E0		*Zero!*
1E2		*Decimal point not required*
	0.1E−12.5	*Exponent must be integer*
	0.1E−123	*Value too small on most computers*
	0.1E+123	*Value too large on most computers*

Data Types
Double Precision Constants

- Whatever applies to real numbers, apply to double precison numbers . . .
- . . . with these additions
  - If 7-digit accuracy is not enough, increase it using double precison numbers
  - Accurate to 14–16 decimal places (machine dependent)
  - Use `D` instead of `E` for exponent, for examples

`0.98153E+12`	`REAL` number
`0.3817253422126D+08`	`DOUBLE PRECISION` number

  - Samples in **Example 2.6**.

EXAMPLE 2.6

The following examples illustrate double precision constants using scientific notation:

Valid Examples	Invalid Examples	Comment
0.0D0		*Double precision form of zero*
0.23D-94		*Double precision will give greater range*
	0.123456789E23	*Not double precision! Extra digits ignored*

Data Types
Complex Constants

- Algebraic representation:

  $\text{real}_1 + (\text{real}_2)i$

  for example, $4 + 3i$, where 4 is the *real* part and $3i$ is the *imaginary* part

- Fortran representation:

  $(\texttt{REAL}_1,\texttt{REAL}_2)$

  where $\texttt{REAL}_1$ is the *real* part and $\texttt{REAL}_2$ is the *imaginary* part

- Samples in **Example 2.7**.

EXAMPLE 2.7

Here are some examples of commonly encountered complex constants:

Valid Examples	Invalid Examples	Comment
(1.23, -3.45)		*Either component may be negative*
(+1.23, 0.0)		*Positive sign is optional*
(1.23E-2, 3.45)		*Exponential format is permitted*
	(1.23D-128, 3.45)	*Both components must match in precision*
	(1, 2)	*Integers not allowed*

Data Types
Character Constants

- Handle non-numeric data such as names and addresses.
- Any set of allowed symbols, defined below, and enclosed in single quote marks (’):
  - Letters of alphabet (upper- and lower-case)
  - Numbers 0 through 9
  - Special characters $+ - () . , * / = ' \$$

Data Types
Character Constants

- Note that
  - '1234' is a character constant but 1234 is its numerical counterpart.
  - You can add 123 + 456 but NOT '123' + '456'.

- Samples in **Example 2.8**.

EXAMPLE 2.8

Here are some commonly encountered examples of character constants:

Valid Examples	Invalid Examples	Comment
'Helen'		*Mixing upper/lower case OK*
'12345'		*All numbers OK*
'I''M OK'		*If you want an apostrophe inside the single quotes, you must use two apostrophes. Result is I'M OK.*
	"Helen"	*Must use single quotes (apostrophe)*
	Helen	*Missing quote marks*
	'I ♥ NY'	*Illegal character (♥)*

Data Types
Logical Constants

- Can only take two values:
  - `.TRUE.`
  - `.FALSE.`

  Note the use of periods!!

- Much used in Fortran control structures, which we will deal later.

- Samples in **Example 2.9**.

EXAMPLE 2.9

Here are some examples of common uses of logical constants:

Valid Examples	Invalid Examples	Comment
.True.		*Mixed case is acceptable*
	FALSE	*Requires periods (.FALSE.)*
	.T.	*Must spell out complete word*

Data Types
Variables

- Variables are
  * means to manipulate data—used to represent a quantity in a formula as used in algebra,
  * also used to represent memory in computer.

- Input and output statements are used to introduce data into program by assigning them to variables

- Work through **Example 2.10**.

EXAMPLE 2.10

Below is a simple program to compute the area and circumference of a circle of radius r. In the program, the variables used are PI, AREA, CIRCUM, and R. Note that we try to choose variable names that indicate their function in the program.

```
      PROGRAM AREAOFCIRCLE
C The following statements request the user to type in
C a value of the radius
      PRINT * , 'Enter circle radius'
      READ * , R
C Once the radius is fed in, the area is calculated
      PI = 3.1416
      AREA = PI * R * R
      CIRCUM = 2 * PI * R
C The value of the area is now printed out
      PRINT * , 'Area of circle is ', AREA
      PRINT * , 'Circumference of circle is ', CIRCUM
      END
```

Data Types
Variables

- Give variables suitable names to describe their function within the program—e.g. `VOLUME`, `AREA`, `WIDTH`, etc.

- Rules for defining Fortran 77 variable names:
  - Names are 1 to 6 characters long
  - Only letters (A–Z), (a–z) and numbers (0–9) allowed
  - First character must be a letter
  - Upper/lower case are equivalent
  - Blank spaces are ignore

- Work through **Example 2.11**.

EXAMPLE 2.11

Here are some common forms of variable names:

Valid Examples	Invalid Examples	Comment
X		*OK, but not very illustrative*
TAXDUE		*Better, since it describes its function*
TEMP1		*OK to mix letters and numbers*
AMT DUE		*OK, spaces are ignored*
Amt Due		*Same as previous example, since lower case is treated the same as upper case in Fortran*
	AMOUNTDUE	*Too many characters (max of 6)*
	$OWED	*Illegal character ($)*
	2BEES	*Must start with a letter*

Data Typing

- How does a program tell the computer to define variables, whether they are integer, real, double precision, complex, character or logical?

- Fortran offers two options:

  1. *implicit* data typing
  2. *explicit* data typing

Data Typing

Implicit Data Typing

- Variable is assigned *data type* based on the first letter of the variable name
- Variable names that begin with letters A–H or O–Z are real. Examples are `RADIUS`, `PI` and `AREA`.
- Variable names that begin with letters I–N are integer. Examples are `ICOUNT`, and `MAXIT`.
- Applicable to integer and real data types only.
- Not applicable to complex, character or logical.
- Work through **Example 2.12**.

EXAMPLE 2.12

Here are some examples of implicit typing:

Variable	Type
R	Real
PI	Real
AREA	Real

Variable	Type
CIRCUM	Real
LENGTH	Integer
ICOUNT	Integer

Data Typing
Explicit Data Typing

- A procedure of specifying *explicitly* how to treat each variable.
- Used to overide the implicit data typing of integer and real variables.
- As implicit data typing is not applicable to complex, character or logical, they must use explicit typing rules.

Data Typing
Explicit Data Typing

- Examples

`REAL X,Y`	declares `X`, `Y` as real variables
`LOGICAL OKEY`	declares `OKEY` as logical variable

- See **Example 2.13** for more examples of declaration.
- See **Example 2.14** on how to apply in a program.

EXAMPLE 2.13

Here are some examples of explicit typing:

Declaration Statement	Result
REAL X , Y , Z	Declares X, Y, and Z as a real variables
REAL LENGTH	Defines LENGTH as a real variable
INTEGER COUNT	Defines COUNT as an integer variable
CHARACTER GRADE	Defines GRADE as a character variable of length 1
CHARACTER*20 NAME	Defines NAME as a character variable of length 20
COMPLEX PHASE	Defines PHASE as a complex variable
LOGICAL YESNO	Defines YESNO as a logical variable
DOUBLE PRECISION X	Defines X as a double precision variable
CHARACTER A*10, B*20	Defines A as a character variable of length 10 and B as a character variable also, but of length 20

EXAMPLE 2.14

The following program is similar to Example 2.10, except that the types of the variables are now explicitly stated:

```
      PROGRAM AREAOFCIRCLE
C The following statements requests the user to type in
C a value of the radius
      REAL R, PI, AREA, CIRCUM
      PRINT * , 'Enter circle radius'
```

(Program continues on next page)

```
      READ * , R
C Once the radius is fed in, the area is calculated
      PI = 3.1416
      AREA = PI * R * R
      CIRCUM = 2 * PI * R
C The value of the area is now printed out
      PRINT * , 'Area of circle is ', AREA
      PRINT * , 'Circumference of circle is ', CIRCUM
      END
```

Simple Input/Output

- Most programs require users to enter data into program. This calls for input statement. To input a value to a variable, from keyboard for instance, we use

```
READ *, variable1, variable2
```

- Once data manipulation is completed, we may want to send results to display. This call for output statement. To output a value of a variable, to the VDU for instance, we use

```
PRINT *, variable1, variable2
```

- Recall I/O in **Example 2.1**. More in **Example 2.15**.

EXAMPLE 2.15

The following example reads in a person's name and age in years. It then converts the age from years into months:

```
      PROGRAM AGEINMONTHS
C The declaration statement must come first
      CHARACTER*10 NAME
      REAL AGEYRS, AGEMTH
C Here is where we input the person's name and age
      PRINT *, 'Enter your name and your age in years'
      READ *, NAME, AGEYRS
C Now we convert the age from years into months
      AGEMTH = AGEYRS * 12
C Print out the results
      PRINT *, NAME, ' is approximately ', AGEMTH, ' months old'
      END
```

Assignment Statements

- Assignment statement is the primary means of storing data in variables. We tell computer to assign a value to a given variable.

- General form is

$$\text{Target} \longleftarrow \text{Value from an expression}$$

- Fortran implementation is

$$\text{Variable} = \text{Value from an expression}$$

Assignment Statements

- Examples of Fortran assignment statements

```
PAY = 5.15
VELOCITY = 45.27
X = SQRT(Y)
ICOUNT = 100
NAME = 'MUHAMMAD IBN ABDULLAH'
OKEY = .TRUE.
```

Assignment Statements: Exception

- Consider conventional algebraic equation

$$x = 1 - x$$

  which, on solving, yields

$$x = \frac{1}{2}$$

Assignment Statements: Exception

- But in Fortran assignment expression

  `X = 1.0 - X`

  has a totally different meaning. It means

  < new value of `X` > = `1.0` - < old value of `X` >

  i.e. take whatever (old) value in a memory location named `X`, substract it from `1.0` and put the result of that calculation back into memory location `X`.

Expressions and Hierarchy of Operations

- There are only *FIVE* basic arithmetic operations in Fortran: subtraction, addition, division, multiplication, exponentiation.

Operation	Fortran symbol	Priority
parentheses	( )	1
exponentiation	**	2
multiplication	*	3
division	/	3
addition	+	4
subtraction	-	4

Priority	Algebraic Symbol	Fortran Symbol	Meaning
1	(.....)	(.....)	Parentheses
2	A^b	**	Exponentiation
3	×	*	Multiplication
3	÷	/	Division
4	+	+	Addition
4	−	−	Subtraction

EXAMPLE 2.21

When two exponentiation operations appear together, they are evaluated right to left:

```
2 ** 3 ** 2      →      2 ** 9      →      512
```

EXAMPLE 2.22

For the examples below, we supply the answer. Trace through each and make sure you get the same result:

Expression	Value	Comments
16.0 − 4.0 − 2.0	10.0	*Left to right*
16.0 − (4.0 − 2.0)	14.0	*Evaluate expression within ( ) first*
16.0 + 4.0 * 2.0	24.0	*Multiplication first*
16.0 / 4.0 / 2.0	2.0	*Left to right*
16.0 ** 4.0 * 2.0	131072.0	*Exponentiation first*
16.0 ** (4.0 * 2.0)	4294967296.0	*Expression within ( ) first*

Name	Description	Argument	Result	Example
ABS(X)	absolute value	integer	integer	J = ABS(−51)
		real	real	X = ABS(−17.3)
		double	double	Z = ABS(−0.1D04)
ACOS(X)	arccosine	real	real (rad)	X = ACOS(0.5)
		double	double (rad)	X = ACOS(0.5D0)
ALOG(X)	natural logarithm	real	real	X = ALOG(2.71828)
		double	double	X = ALOG(0.2718D01)
ALOG10(X)	logarithm base 10	real	real	X = ALOG10(10.0)
		double	double	X = ALOG10(0.1D0)
AMAX(...)	returns largest value	integer	integer	I = AMAX(5,1,6,2)
		real	real	X = AMAX(0.2,5.6)
		double	double	X = AMAX(1D0,3D3)
AMIN(...)	returns smallest value	integer	integer	I = AMIN(4,3,−4)
		real	real	X = AMIN(0.2,5.6)
		double	double	X = AMIN(1D0,3D3)
ASIN(X)	arcsine	real	real (rad)	X = ASIN(0.5)
		double	double (rad)	X = ASIN(0.5D0)
ATAN(X)	arctangent	real	real (rad)	X = ATAN(1.0)
		double	double (rad)	X = ATAN(1.0D0)
COS(X)	cosine	real (rad)	real	X = COS(1.04712)
		double	double	X = COS(1.04712D0)
DBLE(X)	converts to double	integer	double	X = DBLE(3)
		real	double	X = DBLE(3.0)

FORTRAN Intrinsic Functions

Name	Description	Argument	Result	Example
EXP(X)	exponential, e^x	real double	real double	X = EXP(1.0) X = EXP(1.0D0)
INT(X)	converts to integer	real double	integer integer	J = INT(3.9999) J = INT(0.3999D01)
FLOAT(I)	converts to real	integer double	real real	X = FLOAT(4) X = FLOAT(0.4D01)
MOD(I,J)	integer remainder of I/J	integer	integer	J = MOD(29,4)
NINT(X)	round to nearest integer	real double	integer integer	J = NINT(3.99) J = NINT(0.6D01)
REAL(I)	convert to real	integer double	real real	X = REAL(3) X = REAL(0.23D02)
SIN(X)	sine	real (rad) double (rad)	real double	X = SIN(0.5202) X = SIN(0.52D0)
SQRT(X)	square root	real double	real double	X = SQRT(17.6) X = SQRT(0.17D2)
TAN(X)	tangent	real (rad) double	real double	X = TAN(0.785) X = TAN(0.785D0)

FORTRAN Intrinsic Functions

Homeworks

- Go through all **Solved Problems** on pages 59–62.