Computer Science 1510

Lecture 5

Lecture Outline

- Fortran data types
- Fortran variables
- Fortran arithmetic operations
- Fortran assignment statements

Fortran

- Fortran, originally written FORTRAN, is short for FORmula TRANslation.
- Fortran has evolved significantly since its first form in 1957.
- Changes and updates to programming languages result in new versions or standards.
- For example, Fortran 90 is a Fortran standard.
- The rules that specify the behaviour of a programming language standard are set out by a board such as the American National Standards Institute (ANSI).

Basic program structure

- The majority of programming languages have a similar basic structure:
  - A line indicating the start of the program.
  - Declaration statements.
  - Algorithm statements.
  - A line indicating the end of the program.

- Example (Fortran):

```
PROGRAM Print_CS
   WRITE (*,*) 'Computer Science 1510 Lecture notes'
END PROGRAM Print_CS
```

- In the above example we have a Fortran program called `Print_CS` that prints the line

  `Computer Science 1510 Lecture notes`

  to the screen.

Fortran formatting rules

- **Comments**

  It is very important to explain each line or block of code within your source code file. This will allow other users (or even yourself at a later date) to understand your algorithm.

  – All text written after an exclamation mark (!) in a Fortran source file is treated as a comment.
  – Comments are not executed. The compiler knows that it should ignore these portions of the code.
  – A comment can start at the beginning of a line or following a command on a given line.

  Example:

```
PROGRAM Print_CS2
! This program prints a string.
   WRITE (*,*) 'CS 1510' ! String to print
END PROGRAM Print_CS2
```

Fortran formatting rules

- **Line continuation**
  - There is no restriction on the length of a line in a Fortran program.
  - However, it is a good idea to limit the length of the lines to some reasonable value.
  - To continue a long command onto subsequent lines the ampersand character (&) is used at the end of the line to be continued.
  - Can also be used at the beginning of the next line in the case of strings.

  Example:

```
PROGRAM Print_CS3
   WRITE (*,*) 'A string is a series &
                &of characters' ! String to print
END PROGRAM Print_CS3
```

Fortran formatting rules

- **Multiple statements**

  More than one statement can be typed on a single line in a Fortran source file provided that they are separated by semi-colons (;).

```
WRITE (*,*) 'CS'; WRITE (*,*) '1510'
```

- **White space**

  Blank spaces (ie. white space) are ignored except when they occur in the middle of keywords (ex. WRITE). All keywords must NOT contain spaces!

Programming Style

- It is very important to write programs that are readable and understandable.

- Doing so involves adopting a programming style, which may include:
  - Including comments to explain the algorithm where necessary (too many comments can make a program difficult to read).
  - Using meaningful variable names.
  - Indenting blocks of code.
  - Using spaces and blank lines.

- Programs should include a specification block at the beginning of the code which may include:
  - What the program does and how.
  - An explanation of the variables used.
  - The name of the programmer and the date on which it was created and modified.

Data types

- When developing an algorithm, it is important to identify the type of the data involved with solving the problem.

- A *data type* defines the physical representation that a particular form of information will use in computer memory. Recall the different binary representations for integers, real numbers, and characters.

- Fortran intrinsic data types:

INTEGER
REAL
CHARACTER
COMPLEX
LOGICAL

- We will now examine the first three of these data types in more detail.

Integers

- Whole numbers (positive, negative, or zero).
- Negative integers must have a $-$ sign prepended. For non-negative numbers a $+$ sign is optional.
- Examples of integer constants:

  2
  -15
  +634
  1200000
  -3278651

- In Fortran, an integer is a string of digits that does not contain commas or a decimal point.
- 16 or 32 bits.
- The more bits that are allocated for the number, the larger the range of numbers that can be stored.
- Fortran integers: INTEGER

Real numbers

- Floating point or decimal data.
- Can also be written in exponential notation.
- Examples of real constants:

-34.6
3.14159
-10.
15679.0
5.6E-4

- In Fortran, a real number is a string of digits that contains a decimal point. Commas are not permitted.
- 32 or 64 bits.
- Fortran real numbers: REAL
- Note that 10 is an integer while 10.0 is a real number. These are stored differently in memory.

Characters/Strings

- A character is any single letter, number, or symbol.
- A character *string* is a series of characters together, such as a word or sentence (ex. 'today').
- In Fortran, characters are indicated by enclosing them in single or double quotes (ex. 'a', or "a").
- Each character requires one byte of memory (8 bits).
- Recall that a blank space is considered a character, and also requires one byte of memory.
- Fortran characters:

```
CHARACTER
CHARACTER(Len=10)
```

- A quote (or double quote) character can be included in a string by using two consecutive quotes (or double quotes) (ex. 'Don"t').

Identifiers

- An *identifier* is a name used to identify entities such as programs, constants, and variables.

- In Fortran, identifiers must conform to the following rules:

  1. First character must be an alphabetic character (A through Z, uppercase or lowercase).
  2. Maximum of 31 characters.
  3. Characters other than the first may be letters, numbers, or the underscore character (_), but not spaces.

- Note that the underscore character is often used in identifiers to separate words (ex. `num_grades=5`).

- Fortran is not case-sensitive. Variables `GRADE` and `grade` are considered the same variable. However, strings (in quotes) retain their case.

- It is important to use meaningful identifiers!

Fortran variables

- A variable within a computer program, like a mathematical variable, is a symbol that is used to refer to a quantity.

- A variable can have a value assigned to it, and this value can change throughout a program.

- A variable represents a value of some specified type (integer, real, etc.).

- When a variable is used in a program, the compiler associates it with a memory location.

- The value of a variable at any time is the value stored in the associated memory location at that time.

- Variable names are identifiers, and thus must conform to the rules for valid identifiers.

Variable declarations

- The type of a variable determines what type of data can be assigned to its memory location.

- Each variable used should therefore be *declared*, that is, each variable should be assigned a data type.

- Fortran variables can be declared as follows:

  ```
  type-specifier::list
  ```

  where `type-specifier` is any valid Fortran data type (ex. INTEGER), and `list` is a list of variable names (separated by commas) to be declared of that type.

- Variable declarations must be placed at the beginning of a Fortran program, before any executable statements.

Examples: Variable declarations

- Real variables:

```
REAL::sum,average
```

- Integer variables:

```
INTEGER::num,factorial
```

- Character variables:

```
CHARACTER::initial
CHARACTER(Len=20)::first_name
CHARACTER(20)::last_name,letter*1
```

`initial` is a single character
`first_name` is a string of up to 20 characters
`last_name` is a string of up to 20 characters
`letter` is a single character (`*1` overrides the 20)

Fortran statements - IMPLICIT NONE

- Most programming languages require that all variables used in a program be declared.

  Fortran has default types for variables depending on the first character of the identifier (implicit types).

- It is very poor programming practice to rely on such implicit data types.

- Fortran has a statement that will force the programmer to declare all variables used.

  The `IMPLICIT NONE` statement is placed after the `PROGRAM` statement, before any declaration statement.

- When this statement is included, if you attempt to use a variable that has not been declared, the compiler will stop and print an error message.

- You should use the `IMPLICIT NONE` statement in all Fortran programs that you write.

Fortran variable initialization

- In Fortran, all variables are initially undefined.
- It may be the case that certain compilers will initialize variables with numeric types to 0.

  However, this behaviour should not be assumed, and it is good practice to initialize all variables.
- Variable initializations can be done on the declaration line, or within the executable section of the program.
- Example:

```
REAL::sum=0
INTEGER::num
num=5
```

Fortran named constants

- In some programs certain constants may be required and used often. For example, $\pi = 3.14159\ldots$.

- To specify such a constant, the `PARAMETER` attribute can be added to the declaration line.

- Syntax:

  ```
  type-specifier,PARAMETER::list
  ```

  where `type-specifier` and `list` are as described for the variable declarations.

- Example:

  ```
  REAL,PARAMETER::pi=3.14159
  ```

- The value of such named constants cannot change. Any attempt to change the value will result in a compile-time error.

Fortran arithmetic operations

- Basic arithmetic operations in Fortran:

Operator	Function
+	addition
-	subtraction
*	multiplication
/	division
**	exponentiation

- Priority rules for evaluating arithmetic expressions:

  1. Parentheses ( ) from innermost to outermost.
  2. Exponentiation. Consecutive exponentiations are performed right to left.
  3. Multiplication and division. In the order in which they appear from left to right.
  4. Addition and subtraction. In the order in which they appear from left to right.

- Use parentheses for clarity, or when you are in doubt about the priority.

Examples: Order of operations

```
2**3**2
```

```
= 2**9 = 512
```

```
10-8-2
```

```
= 2-2 = 0
```

```
2+4/2
```

```
= 2+2 = 4
```

```
2+4**2/2
```

```
= 2+16/2 = 2+8 = 10
```

```
(5*(11-5)**2)*4+9
```

```
= (5*6**2)*4+9
= (5*36)*4+9
= 180*4+9
= 720+9 = 729
```

Arithmetic operations

- When two constants or variables of the same type are combined using an arithmetic operator (ex. $+$), the result has that same type.

  For example, if we multiply two variables declared as `INTEGER`, the result is an integer.

- It is possible to have arithmetic operations involving constants or variables with different types. Such expressions are referred to as *mixed-mode expressions*.

- When mixed-mode expressions are evaluated, the result is normally converted to the most general type in the expression. E.g., in the case of expressions involving an integer and a real number, the integer is first converted to a real, and the result is a real.

- In general, mixed-mode expressions should be avoided in most instances. (An exception is exponentiation).

Examples: Arithmetic operations

```
3.0 + 8/5
```

```
      = 3.0 + 1 = 3.0 + 1.0 = 4.0
```

```
3 + 8.0/5
```

```
        = 3 + 8.0/5.0 = 3 + 1.6
           = 3.0 + 1.6 = 4.6
```

```
2.0**3
```

```
         =2.0 * 2.0 * 2.0 = 8.0
```

```
(-4.0)**2
```

```
        =(-4.0) * (-4.0) = 16.0
```

```
(-2.0)**3.0
```

$$= e^{3.0\ \ln(-2.0)} = \text{Undefined!}$$

Integer division

- Consider the following Fortran statements:

```
INTEGER::j=2,k=3,m
m=j/k
```

- All three variables, `j`, `k`, and `m` have been declared as integers.

- When dividing `j` by `k` we have two integer variables which will produce an integer.

  However, $2/3 = 0.666\ldots$, which is not an integer. The result is truncation of the answer to give `m`=0.

- In some cases *integer division* produces unwanted results, however, in other cases it can be used to our advantage.

Built-in mathematical functions

- The following is an incomplete list of built-in mathematical functions provided by Fortran:

Function	Description	Arg(s)	Result
ABS(x)	Absolute value of x	I or R	Same
MOD(x,y)	x-INT(x/y)*y	I or R	Same
SQRT(x)	Square root of x	R	R
EXP(x)	Exponential of x	R	R
LOG(x)	Natural log of x (base e)	R	R
LOG10(x)	Log base 10 of x	R	R
SIN(x)	Sine of x (radians)	R	R
COS(x)	Cosine of x (radians)	R	R
TAN(x)	Tangent of x (radians)	R	R
INT(x)	Conversion of x to INTEGER	R	I
REAL(x)	Conversion of x to REAL	I	R

- In some cases, the arguments may also be of type COMPLEX.

Assignment statements

- An *assignment statement* is used to assign values to variables.

- Syntax:

  ```
  variable = expression
  ```

  where `variable` is assigned the value of `expression`.

- The `variable` is any valid Fortran identifier.

- The `expression` can be a constant, a variable that has already been assigned a value, or a formula involving constants and/or variables to be evaluated.

- In the case of a formula, the `expression` is first evaluated, and the resulting value assigned to `variable`.

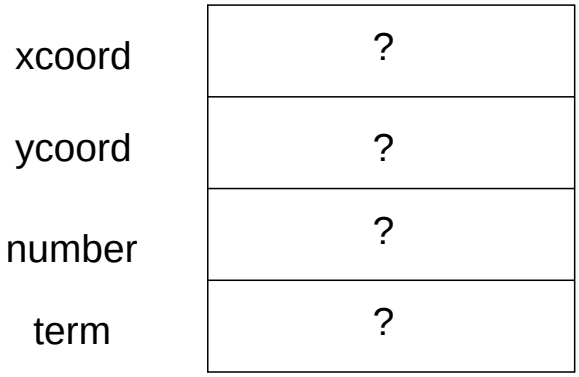
- An assignment statement is NOT a mathematical equation!

Examples: Assignment statements

Consider the following Fortran statements:

```
REAL :: xcoord, ycoord
INTEGER :: number, term
xcoord = 5.23
ycoord = sqrt(25.0)
number = 17
term = number/3 + 2
xcoord = 2.0*xcoord
```

The declaration statements associate locations in memory with the given variable names. Therefore, we would have something like:



where ? indicates that the values of the variables are currently undefined (they could be zero if the compiler initialized them).

Examples: Assignment statements

The three assignment statements following the declarations assign values to `xcoord`, `ycoord`, and `number`, to give:

xcoord	5.23
ycoord	5.0
number	17
term	?

The second last statement assigns a value to `term`. Note the integer division.

xcoord	5.23
ycoord	5.0
number	17
term	7

The final statement replaces the value of `xcoord`.

xcoord	10.46
ycoord	5.0
number	17
term	7

Mixed-mode assignment

- Mixed-mode assignment occurs when the type of `expression` (ie. the result of evaluating the right-hand side of an assignment statement) differs from the type of `variable` (ie. the variable on the left-hand side of the assignment statement).

- If the `expression` evaluates to an integer, but `variable` is a REAL then the integer result is converted to a real number and assigned to the variable.

- If the `expression` evaluates to a real number, but `variable` is an INTEGER then the fractional part of the result is truncated, and the integer part is assigned to the variable.