Computers in Engineering COMP 208

First Look At Fortran

Michael A. Hawker

A First Look At Fortran

- Let's have a look at Fortran
- We will examine a simple program
- How do we get it to run?

Our First Program

```
PROGRAM hello
  IMPLICIT NONE
  !This is my first program

  WRITE (*,*) "Hello World!"

END PROGRAM hello

!Note the use of whitespace (indentation &
!blank lines) to make the program more
!readable
```

How Do I Run The Program?

- First, prepare the program using an editor to enter the program text
  - A plain text editor such as Notepad works, but NOT Word
  - An IDE (Integrated Development Environment) such as SciTE helps layout the program, compile, and run it
- Save the program text with the suffix .f90 (e.g. Hello.f90)

How Do I Run The Program?

- Run the FORTRAN compiler taking its input from this file and producing an executable program
  - If you used a plain text editor, run the following from the command window.
    `gfortran –fimplicit-none –W hello.f90 -o hello.exe`
  - If you used SciTE, you can use the tool bar to compile the program
- Run the executable program (in the .exe file)
  - From the command window, just type “hello”
  - From and IDE like SciTE, choose run from the tool bar

The Program Block

```
PROGRAM hello
  IMPLICIT NONE
  !This is my first program

  WRITE (*,*) “Hello World!“

END PROGRAM hello
```

- The bold keywords tell the compiler where the program begins and ends.
- They bracket a section of code called a block

Some Observations

```
PROGRAM hello
  IMPLICIT NONE
  !This is my first program
  WRITE (*,*) "Hello World!"
END PROGRAM hello
```

- Using uppercase is a convention to distinguish keywords.
- FORTRAN is case insensitive. PROGRAM, program, proGRAM, pRoGrAm are all the same.
- Keywords are **not reserved** in FORTRAN

The Program Block in General

- Syntax for the program block in general looks like:

```
PROGRAM program-name
  IMPLICIT NONE
   {declarations}
   {statements}
END PROGRAM program-name
  {subprogram definitions}
```


A First Program -- Comments

```
PROGRAM hello
  IMPLICIT NONE
  !This is my first program
  WRITE (*,*) "Hello World!"
END PROGRAM hello
```

- Comments are preceded by a "!"
- All characters following the exclamation mark on that line are ignored by the compiler
- The "!" inside the Hello World! string is not part of a comment

Comments

- Comments are used to signal the intent of the programmer
  - Improve readability and understanding
  - An important aid to debugging and maintaining code
- Comments can appear anywhere in the program
- When the compiler encounters a "!" (that is not contained inside a string) it ignores the rest of the line
- Comments are only there for someone reading the program, not for the compiler to use.
- Make Useful Comments

Useful Comments

- Not Useful:

```
! Add 1 to a
a = a + 1
```

- More Useful:

```
! Increment to account for new user login
a = a + 1
```

- Sometimes, Not Necessary:

```
NumUsersLoggedIn = NumUsersLoggedIn + 1
```


A First Program -- Output

```
PROGRAM hello
  IMPLICIT NONE
  !This is my first program
  WRITE (*,*) "Hello World!"
END PROGRAM hello
```

- The WRITE statement instructs the computer to display values on the screen or on some other output device
- The values to be displayed can be strings (as in the example) or any other value (such as a number).

The Write Statement

- The WRITE statement has one of the forms:

```
WRITE (*,*) exp1, exp2, exp3, ...,expn
WRITE (*,*)
```

- The second form outputs a blank line
- The expressions can be of any type
- Each expression is evaluated and the value is displayed on the screen

Controlling Output

- The computer chooses how to display the output on the screen
- We may want to control how the output appears
  - Display monetary numbers with two decimal points
  - Align data in columns
- Later we study FORMAT codes that give us that kind of control
- We also will see how to put the output values into a file or write to some device other than the screen

Let's try solving a real problem

- Here's a classical problem that arises in many applications.
- Problem: Find the roots of the quadratic

$$ax^2+bx+c$$

Roots of a Quadratic

- This problem, and partial solutions are mentioned over 3500 years ago.
- We'll use an algorithm developed in India in the 8th century
- The roots are given by the formula

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The Discriminant

- First we focus on computing the discriminant

b^2-4ac

- We will develop an algorithm for finding the result
- The algorithm should work for any value of a, b and c. That is, it should be generic and robust.

What are a, b and c?

- The values a, b and c are called variables since they can take on any numeric value.
- In Fortran, variables represent memory cells. They are names mapped to memory locations.
- Each cell can store a single value at any given time.
- Each cell's size is dependent on the type of data you want to store there.

Memory "Cells"

A	B	C

How do these values get there?

- Any value stored in a variables (like a, b and c) must be stored in memory
- The value stored can be something you specify beforehand or input from outside the program (user)
- Assignment statements can be used to tell the computer to place values in these cells
- Every time your program runs, the physical memory used by your computer can be different.

What do we do with these values?

- We can use the values stored in variables and perform basic operations such as +, -, *, /, etc. on them
- We can store the result of an operation into a memory cell
- We can output the value to the screen, a file, or a printer

Basic Concept Review

- Algorithms are generic – that is, they must be able to solve the problem in general, not just for some specific values
- We input the values for a specific instance of the problem
- Values are stored in memory cells named by variables
- Algorithms are built using basic operations available on the computer (+, -, *, /)

Algorithm for the Discriminant

- Back to our problem of computing

  b^2-4ac
- A psuedocode algorithm

```
input a, b, c
x <- b * b
y <- a * c
z <- 4 * y
d <- x - z
```


Actions

- Actions to be performed are specified by **statements**
- A basic statement is an **assignment**:

  `x <- y op z`
  - Perform the operation `op` on the values stored in `y` and `z` and then store the result in `x`
- Actions are performed in **sequence**.
- Lines of a program are executed First to Last.
  - The first action is done, then the second, etc…

From pseudocode to FORTRAN

- Each language, including FORTRAN has specific rules for expressing the basic concepts we have discussed
- On the next slide, we look at a FORTRAN version of our discriminant algorithm

The Return of the Discriminant

- Our problem of computing:

  b^2-4ac

- A FORTRAN algorithm

```
READ(*,*) a, b, c
x = b * b
y = a * c
z = 4 * y
d = x - z
```


FORTRAN Variables

- FORTRAN variables are the names of memory cells, programs or functions
- Each name refers to an object of the specified type
- The variable can only hold values of that type
- Declaration statements are used to tell the compiler what variables are to be used in the program

Something New

```
! Compute B*B-4*A*C
PROGRAM Discriminant
  IMPLICIT NONE
  REAL :: a, b, c
  REAL :: d

! read in the coefficients a, b and c

  WRITE(*,*) 'A, B, C Please : '
  READ(*,*) a, b, c

! compute the discriminant d

  d = b*b - 4.0*a*c

! display the results

  WRITE(*,*) 'The discriminant is ', d

END PROGRAM Discriminant
```


Declarations

- Allocate space in memory for a variable
- The size the memory cell will be based on the type of value to be stored
- Create a name for the program to use to refer to that location
- IMPLICIT NONE – Forces Declaration, A Good Thing, Trust Me…

Type Statements

- Declarations are made using type statements
  `type-specifier :: list of names`
- The type-specifier can be
  - `INTEGER`
  - `REAL`
  - `COMPLEX`
  - `LOGICAL`
  - `CHARACTER`
- `INTEGER` variables can hold integer values and `REAL` variables can hold decimal values

Names in FORTRAN

- Computer languages have rules for how to form names
- In FORTRAN, names must start with a letter and can be made up of letters, digits and "_" characters
- It is not safe to use the same name as a FORTRAN keyword
- Create Meaningful Names

User Input

```
! Compute B*B-4*A*C
PROGRAM Discriminant
  IMPLICIT NONE
  REAL :: a, b, c
  REAL :: d

! read in the coefficients a, b and c

  WRITE(*,*) 'A, B, C Please : '
  READ(*,*) a, b, c

! compute the discriminant d

  d = b*b - 4.0*a*c

! display the results

  WRITE(*,*) 'The discriminant is ', d

END PROGRAM Discriminant
```


The READ Statement

- Syntax:
  `READ (*,*) var1, var2, . . ., varn`
- Semantics:
  - Starts a new line to contain the user input
  - Input values must be the same type as the corresponding variables
  - Data must be separated by commas or blanks
  - Extra input values on that line are ignored

The Expression Returns

```
! Compute B*B-4*A*C
PROGRAM Discriminant
  IMPLICIT NONE
  REAL :: a, b, c
  REAL :: d

! read in the coefficients a, b and c

  WRITE(*,*) 'A, B, C Please : '
  READ(*,*) a, b, c

! compute the discriminant d

  d = b*b - 4.0*a*c

! display the results

  WRITE(*,*) 'The discriminant is ', d

END PROGRAM Discriminant
```


Expressions

- We can combine basic operations into more complex expressions

```
REAL :: a, b, c, d
d = b*b - 4*a*c
```

- The computer can still only do one operation at a time
- The compiler breaks this down into basic operations
- Each language has its own rules to determine the sequence of basic actions

Operations

- An arithmetic expression is formed using the operations:
  - \+ (addition)
  - \- (subtraction)
  - * (multiplication)
  - / (division)
  - ** (exponentiation)
- We will discuss these in much more detail in the next lecture.
- Usually the result is stored in another variable

The Final Result…

```
! Compute B*B-4*A*C
PROGRAM Discriminant
  IMPLICIT NONE
  REAL :: a, b, c
  REAL :: d

! read in the coefficients a, b and c

  WRITE(*,*) 'A, B, C Please : '
  READ(*,*) a, b, c

! compute the discriminant d

  d = b*b - 4.0*a*c

! display the results

  WRITE(*,*) 'The discriminant is ', d

END PROGRAM Discriminant
```


Assignment Statement

- The assignment statement has syntax:
  `variable = expression`
- Semantics
  - Evaluates the expression
  - Stores the result in the variable

The Complete Example

```
! Compute B*B-4*A*C
PROGRAM Discriminant
  IMPLICIT NONE
  REAL :: a, b, c
  REAL :: d

! read in the coefficients a, b and c

  WRITE(*,*) 'A, B, C Please : '
  READ(*,*) a, b, c

! compute the discriminant d

  d = b*b - 4.0*a*c

! display the results

  WRITE(*,*) 'The discriminant is ', d

END PROGRAM Discriminant
```

“Old” & New Topics

- Familiar Things
  - Program Block
  - Comments
  - Write Statement
- New Things
  - Declarations
  - Expressions
  - Assignment Statement
  - Read Statement