COMP 208
Computers in Engineering

Lecture 03

Jun Wang
School of Computer Science
McGill University
Fall 2007

Review

- Modern computers use numbers to represent all information, including program instructions
  - ASCII code; machine language code
- Memory
  - divided into cells
  - each cell has a unique numeric address
  - program is loaded into memory when it is executed
- CPU lives in a cycle: fetch-decode-execute
- High-level language
  - easier to write/read/maintain
  - more natural for human programmers
  - free from hardware details (registers, memory addresses)
  - portability
  - must be translated into machine code to execute

A First FORTRAN Program

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

  WRITE (*,*) “Hello World!”

END PROGRAM hello
```

How Do I Run The Program?

1. 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
2. Save the program text with the suffix .f90 or .f (e.g. Hello.f90)

How Do I Run The Program?

3. 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.

```
g77 -x f77 -ffree-form -W hello.f90 -o hello.exe
```

   - If you used SciTE, you can use the tool bar to compile the program

How Do I Run The Program?

4. Run the executable program (in the .exe file)
    - From the command window, just type “hello”
    - From an IDE like SciTE, choose run from the tool bar

`g77/f77` quirks

- If source file doesn’t have the “.f” extension, then `-x f77` must be used to compile the program

```
g77 -x f77 -W hello.f90 -o hello.exe
```

- If source file has the “.f” extension, -x f77 is not necessary.

```
g77 -W hello.f -o hello.exe
```

First program

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

  WRITE (*,*) "Hello World!"

END PROGRAM hello
```

- A program is a list of symbols (tokens)
- keywords (words with special meaning)
  - `PROGRAM, IMPLICIT, NONE, END`
- Identifiers (words created by programmer)
  - `hello, WRITE`
- literals (explicit data values)
  - `"Hello World!"` (character string literal); `123` (integer literal)
- Other
  - ! () * ,

What is a programming language?

- A programming language defines 2 aspects of a program
  - Syntax
    - what words and symbols can be used to write a program, e.g., **`Y1`** valid; **`1Y`** not valid
    - how the words and symbols can be put together to form a valid program, e.g., **`END PROGRAM`** valid; **`PROGRAM END`** not valid
  - Semantics
    - meaning of the arrangement of words/symbols, e.g., `END PROGRAM` marks the end of main program.

FORTRAN 90 program structure

```
PROGRAM program_name
  specification part
  execution part
  internal subprograms
END PROGRAM program_name
  external subprograms
```

- A program is stored in 1 or more files
- A program has 1 main program and 0 or more subprograms

The Program Block

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**
- Program starts with the main program, and terminates when the main program ends.

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

C is case sensitive

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 from the exclamation mark to the end of the line are ignored by the compiler
- The “!” inside the Hello World! string is not part of a comment

C has 2 types of comments:
// and /* */

Comments

- Comments are used to signal the intent of the program
  - improve readability and understanding
  - important aid to debugging and maintaining code
  - required for good programs
- can appear anywhere in the program
- when 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.

Bad Comments

```
PROGRAM hello
  IMPLICIT NONE

  !Change this - too vague!
  WRITE (*,*) “Hello World!”

END PROGRAM hello
```

```
PROGRAM hello
  IMPLICIT NONE

  !Prints Hello World! - too obvious!
  WRITE (*,*) “Hello World!”

END PROGRAM hello
```

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

Example of the WRITE statement

```
PROGRAM hello
  IMPLICIT NONE

  WRITE (*,*) 2007
  WRITE (*,*) “Hello World!”, 2007

END PROGRAM hello
```

```
 2007
 Hello World!2007
```

output

- WRITE can be used to print integers as well as character strings
- WRITE (*,*) means the compiler should decide the output format

Getting data input

- Example task:
  1. get an integer from user
  2. print out the integer on the screen
- We can’t predict what the integer will be
- The solution is to put the integer in memory, and then get it from memory and print it
- We need to refer to that particular piece of memory
- In high-level languages, we use **variables** to refer to data stored in memory

Variable Declarations

- In FORTRAN, we use variables to refer to data stored in memory
- Before we can use a variable, we must declare it -- give it a type and a name
- The type of a variable, once declared, cannot be changed.

Declarations tell the compiler

- To allocate space in memory for a variable
- What “shape” the memory cell should be (i.e. what type of value is to be placed there)
- What name the program will use to refer to that cell

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
- Each variable can be given an initial value

Type declaration examples

```
INTEGER :: day
INTEGER :: month, year
INTEGER :: hour = 15, minute
REAL :: x, y, z
```

The type of a variable determines:

- The range of values it can have
- The operations we can perform on it

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. For instance, **`1Y`** is not a valid name.
- It is not safe and strongly discouraged to use the same name as a FORTRAN keyword

FORTRAN Variables – A Summary

- FORTRAN variables are names of memory cells, programs or functions
- Each name refers to a piece of data of a specific type (we think of as the shape of the cell)
- The cell can only hold values of that shape
- Declaration statements are used to tell the compiler what variables are to be used in the program
- A variable must be declared before it can be used.

Variable examples

```
PROGRAM Var_example
  IMPLICIT NONE
  INTEGER :: year = 2007

  WRITE (*,*) year ! print value of year
END PROGRAM Var_example
```

variable declared
and initialized

```
PROGRAM Var_example
  IMPLICIT NONE
  INTEGER :: year
  year = 2007
  WRITE (*,*) year ! print value of year
END PROGRAM Var_example
```

for an uninitialized variable,
we can use assignment
statement to give it a value

Data input: the READ statement

```
PROGRAM Var_example
  IMPLICIT NONE
  INTEGER :: year

  WRITE (*,*) “Please enter year: ”
  READ *, year
  WRITE (*,*) “The year is: ”, year
END PROGRAM Var_example
```

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 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 focus on computing the discriminant

$\mathtt{b^2-4ac}$

We 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

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 of memory locations.
- Each cell can store a single value at any given time.

How do these values get there?

- The values of variables like a, b and c must be stored in the memory cells
- They can be input from outside the program
- Assignment statements can be used to tell the computer to place values in these cells

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, to a file or to a printer

An algorithm for the discriminant

- Back to our problem of computing

  b^2-4ac

- A pseudo-code algorithm

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

Basic Concepts

- 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 (+, -, *, /)

Actions

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

```
x <- y op z
```

means perform the operation `op` on the values in `y` and `z` and then store the result in `x`

- Actions are performed in **sequence**. The example is a straight line program. The first action is done, then the second and so on.

Expressions

- The computer can only do one operation at a time in the CPU
- To make algorithms easier to express, we can combine operations into more complex expressions
- These expressions must be broken down into a sequence of basic operations
- This is one of the tasks of the compiler in a high level language

Expressions

- We can rewrite the algorithm using more complex expressions

```
input a, b, c
d <- b*b - 4*a*c
```

- The compiler breaks this down into basic operations
- Each language has its own rules to determine the sequence of basic actions

From pseudo-code to FORTRAN

Each language has specific rules for expressing the basic concepts we have discussed

On the next slide, we look at a FORTRAN version of the discriminant algorithm

Some parts of the program look familiar

- Program block
- Comments
- Write statements

```
! --------------------------------------------------
! 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
```