Background

Fortran Language

- Initial development in the 1950’s leading to the release of Fortran I in 1957 for the IBM 704 mainframe.
- One of the earliest high-level alternatives to programming in assembly or machine languages.
- Design goals included machine independence and a (relatively) intuitive syntax. Unfortunately, standardization of the various implementations was not given sufficient importance.
- The lack of standardization led to the development of the first ANSI Fortran Standard in 1966, followed by revised standards adopted in 1978 (Fortran77) and 1992 (Fortran90). The ANSI Standard defines what features a compliant Fortran implementation must have, what syntax the implementation must support for those features, and precisely how those features must behave in (almost) every conceivable situation.
- CS 1014 is based on the Fortran 90 standard.
- Most compiler vendors include additional features, called language extensions, in order to provide their customers with a Fortran implementation that includes language constructs and capabilities that are desirable and/or present in other major languages. Since these language extensions lie outside the official ANSI Standard, different vendors implement them in different ways. The use of such extensions reduces the portability of your source code and is forbidden unless specifically allowed.
Syntax (Grammar)

The syntax of a language defines the vocabulary of the language and how the vocabulary elements (words and punctuation in English) may be combined to form correct phrases and sentences. Fortran (and all programming languages) have precise syntax rules — used by the compiler when it attempts to translate source code into object code.

Semantics (Meaning)

The semantic rules of a language define the meaning of syntactically correct sentences. These include the definitions of the vocabulary elements of the language and, to some degree, rules governing how those vocabulary elements interact and modify each other.

The purpose of many of the slides that follow is to define and illustrate the syntax and semantic rules of the Fortran language. The process of learning a programming language is very similar to that of learning any second language. You will learn to read and write Fortran (it’s not a spoken language).
A Simple Fortran Program

PROGRAM skeleton

! This is a comment line
! These lines declare variable names.
! Since they aren’t used, they are
! really unnecessary.
    IMPLICIT NONE
    INTEGER:: an_int = 100
    REAL:: a_real = 95.25

! This line writes something to the
! screen
    WRITE(*,*) 'Hello, world!
    WRITE(*,*) 'My first Fortran Program Output:'
    WRITE(*,*) an_int, a_real

! These lines stop the program and
! mark its end
    STOP
    END PROGRAM skeleton

If you type this program in, compile it and run it, the output is (something like):

Hello, world!
My First Fortran Program Output:
100    95.2500
Statements

Two types of statements:
1) Executable: causes the computer to perform some action
   ...add two numbers, input some data, transfer control, etc.
2) Specification: does not cause any action; describes some
characteristic of program or data (e.g.: DATA, FORMAT,
   INTEGER)

Variables

Variable name <--> symbolic name for a storage location
1 per storage location
1 - 31 characters (alphanumeric: A-Z, a-z, 0-9, and underscore character _)
1st character must be alphabetic
cannot be a Fortran keyword word

Examples

<table>
<thead>
<tr>
<th>Valid</th>
<th>Invalid</th>
<th>Why invalid?</th>
</tr>
</thead>
<tbody>
<tr>
<td>total</td>
<td>72y</td>
<td></td>
</tr>
<tr>
<td>SUM</td>
<td>3rd</td>
<td></td>
</tr>
<tr>
<td>x34</td>
<td>b2.6</td>
<td></td>
</tr>
<tr>
<td>amount</td>
<td>read</td>
<td></td>
</tr>
<tr>
<td>classgrade</td>
<td>if</td>
<td></td>
</tr>
</tbody>
</table>

CS 1014 Requirements:
- Variable names must be descriptive!! Use TOTAL, not T;
  AMOUNT, not Z, etc.
- Variable name must be longer than 1 or 2 letters!
### Variable Characteristics

#### Internal Storage

<table>
<thead>
<tr>
<th>NAME</th>
<th>VALUE</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>age</td>
<td>[27]</td>
<td>077</td>
</tr>
<tr>
<td>x</td>
<td>[15.3]</td>
<td>07B</td>
</tr>
<tr>
<td>name</td>
<td>['Smith']</td>
<td>07F</td>
</tr>
<tr>
<td>i</td>
<td>[0]</td>
<td>084</td>
</tr>
<tr>
<td>er</td>
<td>[-0.10]</td>
<td>088</td>
</tr>
<tr>
<td>ch</td>
<td>['CHIP']</td>
<td>08C</td>
</tr>
<tr>
<td>letr</td>
<td>['x']</td>
<td>090</td>
</tr>
<tr>
<td>total</td>
<td>[140]</td>
<td>091</td>
</tr>
<tr>
<td>count</td>
<td>[32]</td>
<td>095</td>
</tr>
<tr>
<td>year</td>
<td>[1969]</td>
<td>099</td>
</tr>
</tbody>
</table>

Can use symbolic names for variables.

Computer remembers which memory cells correspond to which variable name.

#### Storage Cells/Locations

- each can hold one item at a time
- each has a unique address (fixed)
- storing a new item in a location causes the old item to be lost (erased)
- at beginning of a program run, each location holds garbage ...
  do not assume that it has blanks, 0’s, etc.

As a programmer, it would be very difficult to keep track of our data items by using these addresses (that the computer must know/use). We avoid using these addresses by using variable names which are available in a

High-level language ---> Fortran
Data Types

Two types of data: Numeric and non-numeric
(i.e. character: later)

Numeric

1) integer (fixed point) no explicit decimal point 24, -2, 6, 0
2) real (floating point) must have a decimal point 2.5, 2.9, -67.1231, 0.0

Storage schemes are different, for example:

1) integer

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 2 & 5 \\
\end{array}
\]

25

2) real

\[
\begin{array}{cccccccc}
0 & 2 & 2 & 5 & 0 & 0 & 0 & 0 \\
\end{array}
\]

25.0 = 0.25 * 10**2

You must declare (explicitly or implicitly) what type of data
is associated with each variable name.

Explicit: use a specification statement

INTEGER :: a, b, num

REAL :: i, j, ave

Specification statements must be placed before the first executable
statement; These statements are called declarations

Implicit: (default) first letter of variable name determines type

i - n ---> integer (CAPS also)

a - h, o - z ---> real (CAPS also)

• Essential Lahey Fortran 90 requires explicit declarations
  of variables
**Fortran Math**

Numeric (integer and real) variables, ex.: i, value, sum

Constants:

- **integer**: 12, 1, -78
- **real**: 3.0, 0.5, -12.9331

Arithmetic expressions

Operands: constants, variables, arithmetic expressions, ...

Operators: **exponentiation**, - **negation**, * multiplication, / division, + addition, - subtraction, ( ) parentheses

Examples:

<table>
<thead>
<tr>
<th>Math</th>
<th>FORTRAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \frac{x}{y} )</td>
<td>( x / y )</td>
</tr>
<tr>
<td>( x^y )</td>
<td>( x ** y )</td>
</tr>
<tr>
<td>( ab )</td>
<td>( a * b )</td>
</tr>
</tbody>
</table>

Syntax:

- `<operand> <operator> <operand> . . .`
  - `a * 2`
  - `k / 3`
  - `k + 2 * b`
  - `(k + 2) * b`
  - `k + (2 * b)`

What is the order of evaluation?

0) **parentheses** (inside out)
1) **exponentiation** - (neg.) (right to left)
2) * multiplication / division (left to right)
3) + addition - (minus) (left to right)
Expression Evaluation

Evaluation Example

\[ \frac{A}{B} * \frac{C}{B} - D + X ** 2 * 2 + E \]

\[ 2 ** 3 ** 2 = 512 = 2 ** (3 ** 2) \]
\[ (2 ** 3) ** 2 = 64 \]
\[ -4 ** 2 = -16 = -4 ** 2 \]
\[ (-4) ** 2 = 16 = (-4) ** 2 \]
\[ 4 (A+B) (C+D) = 4 * (A + B) * (C + D) \]

Other Examples
Expression Conversion

Examples:

**FORTRAN ---> Algebraic expressions**

1) \(3.7 - a \times 2.5 / c + d ** 2 + e\)
   \[3.7 - \frac{a (2.5)}{c} + d^2 + e\]

2) \(3.7 - a \times 2.5 / (c + d ** 2) + e\)
   \[3.7 - \frac{a (2.5)}{c + d} + e\]

3) \(3.7 - a \times (2.5 / (c + d ** 2) + e)\)
   \[3.7 - a \left(\frac{2.5}{c + d^2} + e\right)\]

**Examples: Algebraic expressions ---> Fortran**

1) \(\frac{a + b}{c + d}\)
   \[\frac{a + b}{c + d}\]  Wrong
   \[(a + b) / (c + d)\]  Right

2) \(\frac{a}{bc}\)
   \[\frac{a}{b * c}\]  Wrong
   \[a / (b * c)\]  Right
Fortran Program Skeleton

Program Outline

PROGRAM kitchen_floor
  INTEGER:: ...
  REAL:: ...
  OPEN ( 9, FILE = 'p1.dat' )
  OPEN ( 10, FILE = 'p1.rst' )
  READ ( 9, ... ) ...
  WRITE ( 10, ... ) ...
  CLOSE (9)
  CLOSE (10)
END PROGRAM kitchen_floor

Note: File number coordination
  9 -> input
  10 -> output
Arithmetic Processing

Integer mode:
If BOTH operands are integer, then the result is integer (truncate, if necessary, do NOT round off)

\[
\begin{align*}
7 / 2 & \longrightarrow 3 \\
2 / 15 & \longrightarrow 0 \\
8 / 3 & \longrightarrow 2
\end{align*}
\]

Assume
\[
\text{INTEGER:: i, j}
\]
\[
i = 7
\]
\[
j = 2
\]
Then
\[
i / j \longrightarrow 3
\]

Mixed mode:
If one (or both) operand(s) is/are real, then the result is real.

\[
\begin{align*}
7.0 / 2 & \longrightarrow 3.5 \\
2 / 5.0 & \longrightarrow 0.4 \\
9.0 / 3.0 & \longrightarrow 3.0 \\
7.0 * 2 + 5 & \longrightarrow 19.0 \\
7 / 2 * 10.0 & \longrightarrow 30.0
\end{align*}
\]

Avoid Mixed-Mode Arithmetic !!!
Assignment Statement

Syntax: \(<variable> = <expression>\)

Evaluate right hand side first, then store the result in the variable on the left hand side.

Examples: Consider the declarations

\[
\text{INTEGER:: ind} \\
\text{REAL:: total}
\]

<table>
<thead>
<tr>
<th>assignment</th>
<th>result in memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>ind = 7 / 3</td>
<td>2</td>
</tr>
<tr>
<td>ind = ind + 1</td>
<td>3</td>
</tr>
<tr>
<td>total</td>
<td>48.3</td>
</tr>
<tr>
<td>total = total * 2.0</td>
<td>96.6</td>
</tr>
<tr>
<td>total = total * 2</td>
<td>193.2</td>
</tr>
</tbody>
</table>

You can have mixed mode assignment, BUT ...

\[
\text{integervariable = realvalue TRUNCATION} \\
\text{realvariable = integervalue FLOAT}
\]

Examples:

<table>
<thead>
<tr>
<th>total</th>
<th>ind</th>
</tr>
</thead>
<tbody>
<tr>
<td>48.3</td>
<td>96</td>
</tr>
<tr>
<td>total</td>
<td>ind</td>
</tr>
<tr>
<td>3.0</td>
<td>7</td>
</tr>
</tbody>
</table>

Use INT and REAL (FORTRAN built-in or intrinsic functions)

\[
\text{ind = INT(total) * 2 (integer)} \\
\text{total = REAL(ind) / 2.0 (real)}
\]

more on this later!