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 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.

Fortran Language Elements

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 & Variables

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

```
Valid            Invalid      Why invalid?
-------------------------
total            72y
SUM              3rd
x34              b2.6
amount           read
classgrade       if
```

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

	NAME	VALUE	ADDRESS
	age	[27]	077
	x	[15.3]	07B
Can use symbolic	name	['Smith']	07F
names for variables.	i	[0]	084
	er	[-0.10]	088
Computer remembers which	ch	['CHIP']	08C
memory cells correspond	letr	['x']	090
to which variable name.	total	[140]	091
	count	[32]	095
	year	[1969]	099
		(location)	

Storage Cells/Locations

- each can hold <u>one</u> 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	0 0 0 0 0 0 2 5	25
2) real	0 2 2 5 0 0 0 0	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 execuatble 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

Arithmetic Processing

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:

Math		FORTRAN
$\frac{x}{y}$	--->	x / y
x^y	--->	x ** y
ab	--->	a * b

Syntax:

<operand> <operator> <operand> . . .

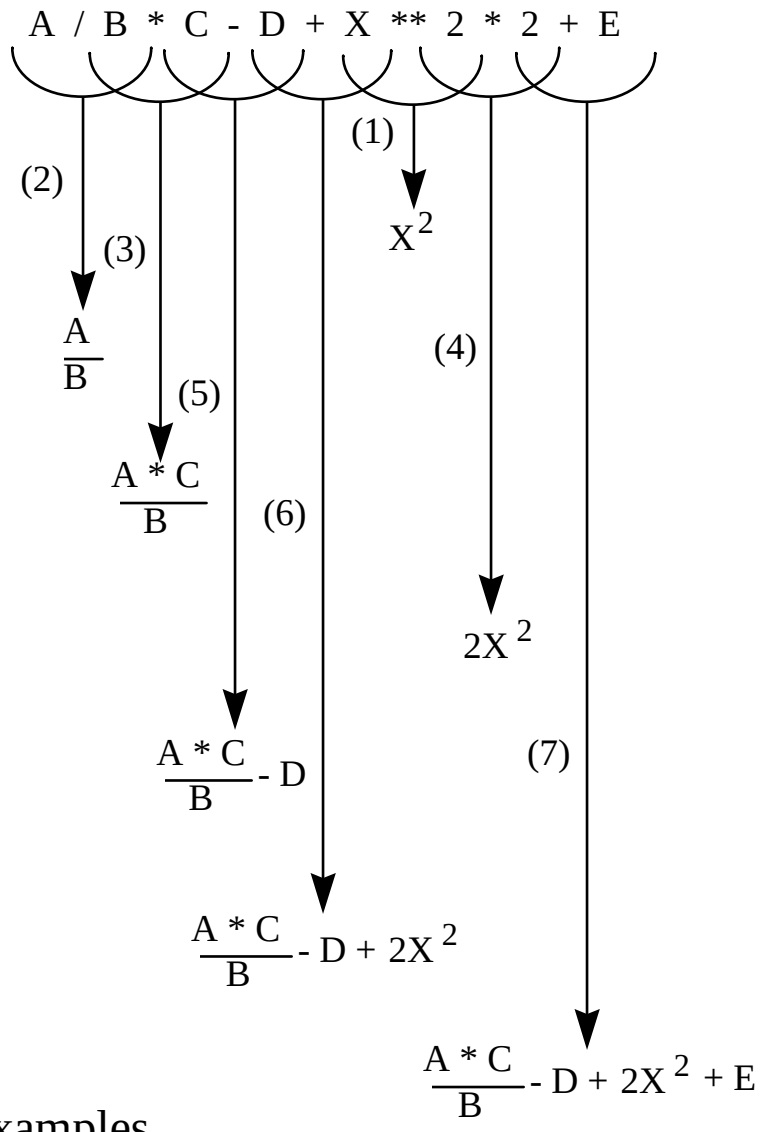
```
a * 2
k / 3
k + 2 * b
(k + 2 ) * b
k + (2 * b)
```

What is the order of evaluation ?

0)	parentheses		(inside out)
1)	**	- (neg.)	(right to left)
2)	*	/	(left to right)
3)	+	- (minus)	(left to right)

Expression Evaluation

Evaluation Example



Other Examples

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)

Expression Conversion

Examples:

FORTRAN ---> Algebraic expressions

1) 3.7 - a * 2.5 / c + d ** 2 + e

$$3.7 - \frac{a\,(2.5)}{c} + d^{2} + e$$

2) 3.7 - a * 2.5 / (c + d ** 2) + e

$$3.7 - \frac{a\,(2.5)}{c + d^{2}} + e$$

3) 3.7 - a * (2.5 / (c + d ** 2) + e)

$$3.7 - a\left(\frac{2.5}{c + d^{2}} + e\right)$$

Examples: Algebraic expressions ---> Fortran

1) $\frac{a + b}{c + d}$

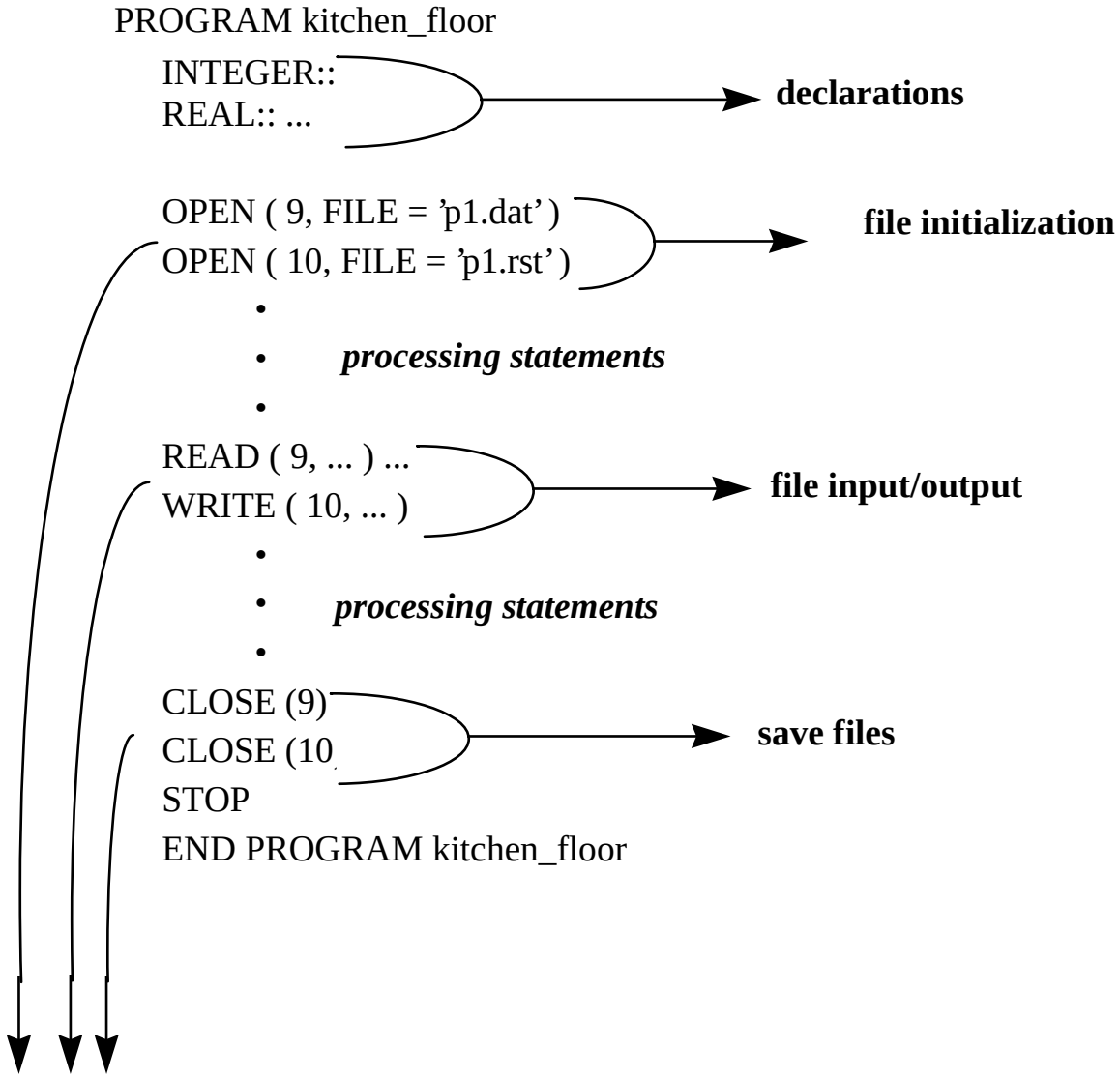
a + b / c + d	Wrong
(a + b) / (c + d)	Right

2) $\frac{a}{bc}$

a / b * c	Wrong
a / (b * c)	Right

Fortran Program Skeleton

Program Outline



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)

```
 7 / 2        --->   3
2 / 15        --->   0
8 / 3         --->   2
Assume
   INTEGER:: i, j
   i = 7
   j = 2
Then
   i / j --->   3
```

Mixed mode:

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

```
 7.0 / 2        ------------>  3.5
 2 / 5.0        ------------>  0.4
9.0 / 3.0       ------------>  3.0
7.0 * 2 + 5     ------------>  19.0
7 / 2 * 10.0    ------------>  30.0
```

Avoid Mixed-Mode Arithmetic !!!

Memory Characteristics

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

INTEGER:: ind

REAL:: total

assignment	result in memory
	ind
ind = 7 / 3	2
ind = ind + 1	3
	total
	48.3
total = total * 2.0	96.6
total = total * 2	193.2

You can have mixed mode assignment, BUT ...

integervariable	= realvalue	TRUNCATION
realvariable	= integervalue	FLOAT

Examples:

	total	ind
ind = total * 2	48.3	96
	total	ind
total = ind / 2	3.0	7

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

ind = INT(total) * 2 (integer)

total = REAL(ind) / 2.0 (real)

more on this later !