Variables Outline

See *Programming in Fortran 90/95*, 1st or 2nd ed, Chapter 5 as well as Chapter 6 sections 6.2 – 6.5.

Fortran 90 Character Set

These are the characters that Fortran 90 recognizes.

- Letters

```
A  B  C  D  E  F  G  H  I  J  K  L  M
N  O  P  Q  R  S  T  U  V  W  X  Y  Z
a  b  c  d  e  f  g  h  i  j  k  l  m
n  o  p  q  r  s  t  u  v  w  x  y  z
```

- Digits

```
0  1  2  3  4  5  6  7  8  9
```

- Special Characters

  space (also known as *blank*)

```
'  "  (  )  *  +  -  /  :  =  _
!  &  $  ;  <  >  %  ?  ,  .
```

Fortran 90 is *case insensitive*: it doesn't distinguish between upper case and lower case letters, except inside character strings. So this program:

```
PROGRAM hello_world
    IMPLICIT NONE
    PRINT *, "Hello, world!"
END PROGRAM hello_world
```

is treated the same as this program:

```
program hello_WORLD
    iMpLiCiT NoNE
    prinT *, "Hello, world!"
end proGRAM HELLO_world
```

Note: a *character string literal* is a sequence of characters *delimited* either by a double quote at the beginning and at the end, or by a single quote at the beginning and at the end; the two delimiters of a string must both be the same quote character.

Basic Data Types

- Numeric
  - `INTEGER`
  - `REAL`
- Non-numeric
  - `LOGICAL`
  - `CHARACTER`

```
PROGRAM basictypes
  IMPLICIT NONE
  INTEGER :: count, number_of_silly_people
  REAL    :: standard_deviation, relative_humidity
  LOGICAL :: count_is_less_than_5, I_am_Henry
  CHARACTER (LEN = 20) :: username, hometown
END PROGRAM basictypes
```


Variables

A *variable* is an association between

- a name (chosen by the programmer), and
- a location in memory (chosen by the compiler).

Every variable has:

- a *name*, chosen by the programmer;
- an *address* (i.e., a location in memory), chosen by the compiler;
- a *data type* (e.g., `INTEGER`, `REAL`, `LOGICAL`, `CHARACTER`), chosen by the programmer;
- a *value* (which may be undefined), sometimes chosen by the programmer, and sometimes determined while the program is running. The value is sometimes called the *contents* of the variable — that is, the value is the contents of the variable's memory location.

The value of a variable can be changed at runtime. We'll see how in a moment.

Jargon: *compile time* and *runtime*.

- Events that occur while a program is being compiled are said to happen at *compile time*.
- Events that occur while a program is running are said to happen at *runtime*.

For example, the address of a variable is chosen at compile time, while its value typically is determined at runtime.

Fortran 90 Variable Declaration

```
INTEGER ::  x
```

This *declaration* tells the compiler to choose a location in memory, name it x, and think of it as an integer. Note that this declaration doesn't specify a value.

The compiler might decide that x will live at, say, address 3980 or address 98234092 or address 56436. We don't know and don't care what address x lives at, because the compiler will take care of that for us; it's enough to know that x has an address and that the address will stay the same throughout a given run of the program.

x : ???????? (Address 56436)

When x is first declared, we don't know what its value is, because we haven't put anything into its memory location yet, so we say that its value is *undefined*, or, informally, *garbage*.

Note: some compilers for some languages automatically initialize newly declared variables to default values (e.g., all integers get initialized to zero), but not every compiler does automatic initialization.

You should **NEVER NEVER NEVER** assume that the compiler will initialize your variables for you.

You should **ALWAYS ALWAYS ALWAYS** explicitly assign values to your variables in the body of the program.

Variable Assignment

An *assignment* statement sets the contents of a specified variable to a specified value:

```
x = 5
```

This statement tells the compiler to put the integer value 5 into the memory location named x, like so:

x : 5 (Address 56436)

So, for example, we might have:

```
INTEGER ::  x
```

⇓

x : ???????? (Address 56436)

```
x = 5
```

⇓

x : 5 (Address 56436)

```
x = 12
```

⇓

x : 12 (Address 56436)

We say "x is assigned twelve" or "x gets twelve."

Variable Assignment Example

```
% cat assign.f90
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!! Program:  assign                                !!!
!!! Author:   Henry Neeman (hneeman@ou.edu)     !!!
!!! Course:   CS 1313 010 Spring 2003           !!!
!!! Lab:      Sec 012 Fridays 1:30pm            !!!
!!! Description:  Declares, assigns and         !!!
!!!   outputs a variable.                       !!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
PROGRAM assign
!
! ***********************************************
! * Declaration section                          *
! ***********************************************
!
! * All variables must be explicitly declared.
!
   IMPLICIT NONE
!
! ******************
! * Local variables *
! ******************
!
! * height_in_cm:  my height in cm
!
   INTEGER :: height_in_cm
!
! ***********************************************
! * Execution section                            *
! ***********************************************
!
! * Assign the integer value 160 to height_in_cm.
!
   height_in_cm = 160
!
! * Print height_in_cm to standard output.
!
   PRINT *, "My height is ", height_in_cm, " cm."
END PROGRAM assign
% f95 -o assign assign.f90
% assign
 My height is  160  cm.
```


Changing a Variable's Contents

```
% cat change.f90
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!! Program:  change                                !!!
!!! Author:   Henry Neeman (hneeman@ou.edu)     !!!
!!! Course:   CS 1313 010 Spring 2003           !!!
!!! Lab:      Sec 012 Fridays 1:30pm            !!!
!!! Description:  Declares, assigns, changes    !!!
!!!   and outputs a variable.                   !!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
PROGRAM change
!
! ***********************************************
! * Declaration section                          *
! ***********************************************
!
! * All variables must be explicitly declared.
!
   IMPLICIT NONE
!
! ******************
! * Local variables *
! ******************
!
! * height_in_cm:  my height in cm
!
   INTEGER :: height_in_cm
!
! ***********************************************
! * Execution section                            *
! ***********************************************
!
! * Assign the integer value 160 to height_in_cm.
!
   height_in_cm = 160
!
! * Print height_in_cm to standard output.
!
   PRINT *, "My height is ", height_in_cm, " cm."
!
! * Assign the integer value 200 to height_in_cm.
!
   height_in_cm = 200
!
! * Print height_in_cm to standard output.
!
   PRINT *, "My height is ", height_in_cm, " cm."
END PROGRAM change
% f95 -o change change.f90
% change
 My height is  160  cm.
 My height is  200  cm.
```

Variable Initialization

We can *initialize* a variable's value in the variable declaration:

```
INTEGER ::  x = 5
```

This statement is the same as declaring `x` and then assigning it `5`.

```
% cat initialize.f90
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!! Program:  initialize                          !!!
!!! Author:   Henry Neeman (hneeman@ou.edu)       !!!
!!! Course:   CS 1313 010 Spring 2003             !!!
!!! Lab:      Sec 012 Fridays 1:30pm              !!!
!!! Description:  Declares/initializes and        !!!
!!!   outputs a variable.                         !!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
PROGRAM initialize
!
! ***********************************************
! * Declaration section                         *
! ***********************************************
!
! * All variables must be explicitly declared.
!
   IMPLICIT NONE
!
! ******************
! * Local variables *
! ******************
!
! * height_in_cm:  my height in cm
!
   INTEGER :: height_in_cm = 160
!
! ***********************************************
! * Execution section                           *
! ***********************************************
!
! * Print height_in_cm to standard output.
!
   PRINT *, "My height is ", height_in_cm, " cm."
END PROGRAM initialize
% f95 -o initialize initialize.f90
% initialize
 My height is  160  cm.
```


Fortran 90 Variable Names

Fortran 90 *symbolic names* (including variable names), which are also called *identifiers*, have the following properties:

- Constructed using only these characters:
  - Letters (case insensitive)

    `a b c d e f g h i j k l m`
    `n o p q r s t u v w x y z`
  - Digits

    `0 1 2 3 4 5 6 7 8 9`
  - Underscore (**NOTE: NOT hyphen)**

    `_`
- At most 31 characters long:
  `name5678901234567890123456789 01`
  is good, but not
  `name56789012345678901234567890 12`
  (Note: not all F90 compilers strictly enforce this property.)
- The first character is a letter:
  `a123_456` is good, but not `1a23_456` nor `_a123_456`

So, since there are 26 letters, 10 digits and 1 underscore, the number of possible variable names is:

$$\sum_{i=0}^{30} 26 \cdot 37^i \approx 3 \times 10^{48}$$

Rule of Thumb for Choosing Variable Names

A variable name should be so obvious that your favorite professor in your major, even if they know nothing about programming, could immediately tell what the variable name means.

Implicitly Declared Variables Are BAD BAD BAD

Most versions of Fortran, including Fortran 90, support *implicitly declared variables* (also known as *implicitly typed variables*): if a variable is not explicitly declared like so

```
INTEGER ::  x
```

then it will be implicitly declared the first time that it appears in the program, and its type will depend on the first letter of its variable name:

- If the first letter of the variable name is one of
  `i  j  k  l  m  n`
  then the variable will be implicitly declared as an `INTEGER`.
- Otherwise, the variable will be implicitly declared as a `REAL`.

However, Fortran 90 provides a statement

```
IMPLICIT NONE
```

that you can put at the beginning of your declaration section (i.e., right after the `PROGRAM` statement) that eliminates implicit variable declaration.

If your program has an `IMPLICIT NONE` statement, then any variables that are not explicitly declared in the declaration section are assumed to be bugs.

If you fail to use the `IMPLICIT NONE` statement, you can have all kinds of problems ...

WHY Are Implicitly Declared Variables BAD BAD BAD?

Here's a little program that does some calculating, then stops to figure out whether the company has gone bankrupt.

```
% cat ruin.f90
PROGRAM ruin
    INTEGER :: bankrupt = 3

    bankrpt = bankrupt - 3
    IF (bankrupt > 0) THEN
        PRINT *, "Bankrupt!"
    ELSE
        PRINT *, "Whew!"
    END IF
END PROGRAM ruin
% f95 -o ruin ruin.f90
% ruin
 Bankrupt!
```

Uh oh. The company has gone bankrupt!

Notice that the program above doesn't have an `IMPLICIT NONE` statement, so the compiler assumes that `bankrpt` is a new variable that we didn't bother to declare, rather than a typo.

Why Is `IMPLICIT NONE` Good?

If we put an `IMPLICIT NONE` statement at the beginning of our declaration section, then the compiler will catch the typo:

```
% cat dontruin.f90
PROGRAM ruin
    IMPLICIT NONE !!! <--- LOOK! LOOK! LOOK!
    INTEGER :: bankrupt = 3

    bankrpt = bankrupt - 3
    IF (bankrupt > 0) THEN
        PRINT *, "Bankrupt!"
    ELSE
        PRINT *, "Whew!"
    END IF
END PROGRAM ruin
% f95 -o dontruin dontruin.f90
Error: dontruin.f90, line 5: Implicit type
       for BANKRPT detected at BANKRPT@=
[f95 terminated - errors found by pass 1]
```

So we **ALWAYS ALWAYS ALWAYS** use `IMPLICIT NONE`, which turns off implicit variable declaration.

Literal Constants

A *constant* is a value that cannot change.

A *literal constant* is a constant whose value is specified literally:

- `INTEGER` literal constants
  (e.g., `5, 0, -127, 403298, -385092809`)
- `REAL` literal constants
  (e.g., `5.2, 0.0, -127.5, 403298.2348, -3.85092809E+08`)
- `LOGICAL` literal constants
  (e.g., `.TRUE., .FALSE.`)
- `CHARACTER` literal constants
  (e.g., `"A", 'Henry', "What's it to ya?"`)
  Note: `CHARACTER` literal constants are sometimes called *string* literal constants, or just *string literals* for short.

Example:

```
% cat tax1997_literal.f90
PROGRAM tax1997_literal
    IMPLICIT NONE
    REAL :: income, tax
    PRINT *, "I'm going to calculate the ", &
 &           "federal income"
    PRINT *, "  tax on your 1997 income."
    PRINT *, "What was your 1997 income in dollars?"
    READ *, income
    tax = (income - (4150.0 + 2650.0)) * 0.15
    PRINT "(A,A,F9.2)", "The 1997 federal income ", &
 &                      "tax on $", income
    PRINT "(A,F8.2,A)", "  was $", tax, "."
END PROGRAM tax1997_literal
% f95 -o tax1997_literal tax1997_literal.f90
% tax1997_literal
 I'm going to calculate the federal income
   tax on your 1997 income.
 What was your 1997 income in dollars?
 20000
The 1997 federal income tax on $ 20000.00
  was $ 1980.00.
```

Named Constants

A *named constant* is exactly like a variable except that its value is set at compile time and **CANNOT** change at runtime. In its declaration, we indicate that it's a constant via the `PARAMETER` *attribute*, and we initialize it:

```
      REAL,PARAMETER ::  PI = 3.1415926
```

```
% cat circlecalc.f90
PROGRAM circle_calculation
    IMPLICIT NONE
    REAL,PARAMETER :: pi = 3.1415926
    REAL,PARAMETER :: diameter_factor = 2.0
    REAL,PARAMETER :: area_power = 2.0
    REAL :: radius, circumference, area
    PRINT *, "I'm going to calculate a circle's"
    PRINT *, "  circumference and area."
    PRINT *, "What's the radius of the circle?"
    READ *, radius
    circumference = pi * radius * diameter_factor
    area = pi * radius ** area_power
    PRINT *, "The circumference is ", circumference
    PRINT *, "  and the area is ", area, "."
END PROGRAM circle_calculation
% f95 -o circlecalc circlecalc.f90
% circlecalc
 I'm going to calculate a circle's
   circumference and area.
 What's the radius of the circle?
 5
 The circumference is   31.4159241
   and the area is   78.5398102 .
```

Named Constant's Value Can't Be Changed

```
% cat paramassign.f90
PROGRAM paramassign
    IMPLICIT NONE
    REAL,PARAMETER :: pi = 3.1415926
    pi = 3.0
END PROGRAM paramassign
% f95 -o paramassign paramassign.f90
Error: paramassign.f90, line 4: Inappropriate use
       of symbol PI detected at PI@=
Error: paramassign.f90, line 4: Attempt to set the
       value of PARAMETER PI detected at PI@=
[f95 terminated - errors found by pass 1]
```


Why Literal Constants Are BAD BAD BAD

When you embed literal constants in the body of your program, you make it **much** harder to *maintain* and *upgrade* your program.

```
% cat tax1997_literal.f90
PROGRAM tax1997_literal
    IMPLICIT NONE
    REAL :: income, tax
    PRINT *, "I'm going to calculate the ", &
 &                   "federal income"
    PRINT *, "  tax on your 1997 income."
    PRINT *, "What was your 1997 income in dollars?"
    READ *, income
    tax = (income - (4150.0 + 2650.0)) * 0.15
    PRINT "(A,A,F9.2)", "The 1997 federal income ", &
 &                      "tax on $", income
    PRINT "(A,F8.2,A)", "  was $", tax, "."
END PROGRAM tax1997_literal
% f95 -o tax1997_literal tax1997_literal.f90
% tax1997_literal
 I'm going to calculate the federal income
   tax on your 1997 income.
 What was your 1997 income in dollars?
 20000
The 1997 federal income tax on $ 20000.00
  was $ 1980.00.
```

```
% cat tax1999_literal.f90
PROGRAM tax1999_literal
    IMPLICIT NONE
    REAL :: income, tax
    PRINT *, "I'm going to calculate the ", &
 &                   "federal income"
    PRINT *, "  tax on your 1999 income."
    PRINT *, "What was your 1999 income in dollars?"
    READ *, income
    tax = (income - (4300.0 + 2750.0)) * 0.15
    PRINT "(A,A,F9.2)", "The 1999 federal income ", &
 &                      "tax on $", income
    PRINT "(A,F8.2,A)", "  was $", tax, "."
END PROGRAM tax1999_literal
% f95 -o tax1999_literal tax1999_literal.f90
% tax1999_literal
 I'm going to calculate the federal income
   tax on your 1999 income.
 What was your 1999 income in dollars?
 20000
The 1999 federal income tax on $ 20000.00
  was $ 1942.50.
```

Why Named Constants Are Good

When you used named constants in the body of your program instead of literal constants, you isolate the constant values in the declaration section, making them trivial to find and to change.

```
% cat tax1997_named.f90
PROGRAM tax1997_named
    IMPLICIT NONE
    REAL,PARAMETER    :: standard_deduction = 4150.0
    REAL,PARAMETER    :: single_exemption   = 2650.0
    REAL,PARAMETER    :: tax_rate           = 0.15
    INTEGER,PARAMETER :: tax_year = 1997
    REAL :: income, tax
    PRINT *, "I'm going to calculate the federal income"
    PRINT *, "  tax on your ", tax_year, " income."
    PRINT *, "What was your ", tax_year, &
 &           " income in dollars?"
    READ *, income
    tax = (income - &
 &          (standard_deduction + single_exemption)) * tax_rate
    PRINT "(A,I4,A,F9.2,A,F8.2,A)",                              &
 &      "The ", tax_year, " federal income tax on $", income, &
 &      "  was $", tax, "."
END PROGRAM tax1997_named
% f95 -o tax1997_named tax1997_named.f90
% tax1997_named
 I'm going to calculate the federal income
   tax on your  1997  income.
 What was your  1997  income in dollars?
 20000
The 1997 federal income tax on $ 20000.00  was $ 1980.00.
```

```
% cat tax1999_named.f90
PROGRAM tax1999_named
    IMPLICIT NONE
    REAL,PARAMETER    :: standard_deduction = 4300.0
    REAL,PARAMETER    :: single_exemption   = 2750.0
    REAL,PARAMETER    :: tax_rate           = 0.15
    INTEGER,PARAMETER :: tax_year = 1999
    REAL :: income, tax
    PRINT *, "I'm going to calculate the federal income"
    PRINT *, "  tax on your ", tax_year, " income."
    PRINT *, "What was your ", tax_year, &
 &           " income in dollars?"
    READ *, income
    tax = (income - &
 &          (standard_deduction + single_exemption)) * tax_rate
    PRINT "(A,I4,A,F9.2,A,F8.2,A)",                              &
 &      "The ", tax_year, " federal income tax on $", income, &
 &      "  was $", tax, "."
END PROGRAM tax1999_named
% f95 -o tax1999_named tax1999_named.f90
% tax1999_named
 I'm going to calculate the federal income
   tax on your  1999  income.
 What was your  1999  income in dollars?
 20000
The 1999 federal income tax on $ 20000.00  was $ 1942.50.
```


&: the Fortran 90 Continuation Character

In Fortran 90, each statement is supposed to take only one line of text, and only one statement can appear on each line of text.

As a general rule, in Fortran 90 we like our lines to run no longer than 72 characters, both for historical reasons — previous versions of Fortran (e.g., Fortran 77) **required** that lines be no longer than 72 characters — and to keep our programs more readable on standard terminals and printers, which typically are 80 characters wide (we like to leave a margin of error).

However, quite often it happens that we have a statement that is too long to fit on a single line.

Fortran 90 provides a continuation character, & (called "ampersand"), that allows statements to be longer than a single line.

Fortran 90 Continuation Character Example

```
% cat tax1997_named.f90
PROGRAM tax1997_named
    IMPLICIT NONE
    REAL,PARAMETER    :: standard_deduction = 4150.0
    REAL,PARAMETER    :: single_exemption   = 2650.0
    REAL,PARAMETER    :: tax_rate           = 0.15
    INTEGER,PARAMETER :: tax_year = 1997
    REAL :: income, tax
    PRINT *, "I'm going to calculate the federal income"
    PRINT *, "  tax on your ", tax_year, " income."
    PRINT *, "What was your ", tax_year, &
 &           " income in dollars?"
    READ *, income
    tax = (income - &
 &         (standard_deduction + single_exemption)) * tax_rate
    PRINT "(A,I4,A,F9.2,A,F8.2,A)",                          &
 &      "The ", tax_year, " federal income tax on $", income, &
 &      "  was $", tax, "."
END PROGRAM tax1997_named
% f95 -o tax1997_named tax1997_named.f90
% tax1997_named
 I'm going to calculate the federal income
   tax on your  1997  income.
 What was your  1997  income in dollars?
 20000
The 1997 federal income tax on $ 20000.00  was $ 1980.00.
```

Notice that a continuation is indicated by an ampersand at the end of a line **as well as** at the beginning of the next line. It isn't strictly necessary to have the ampersands line up nicely, but it makes the program more readable.

List-Directed Output

In Fortran 90, we can print out multiple pieces of information on a single line of output text, using *list-directed output*:

```
% cat circlecalc.f90
PROGRAM circle_calculation
    IMPLICIT NONE
    REAL,PARAMETER :: pi = 3.1415926
    REAL,PARAMETER :: diameter_factor = 2.0
    REAL,PARAMETER :: area_power = 2.0
    REAL :: radius, circumference, area
    PRINT *, "I'm going to calculate a circle's"
    PRINT *, "  circumference and area."
    PRINT *, "What's the radius of the circle?"
    READ *, radius
    circumference = pi * radius * diameter_factor
    area = pi * radius ** area_power
    PRINT *, "The circumference is ", circumference
    PRINT *, "  and the area is ", area, "."
END PROGRAM circle_calculation
% f95 -o circlecalc circlecalc.f90
% circlecalc
 I'm going to calculate a circle's
   circumference and area.
 What's the radius of the circle?
 5
 The circumference is   31.4159241
   and the area is   78.5398102 .
```

Look at the last `PRINT` statement:

```
PRINT *, " and the area is ", area, "."
```

This `PRINT` statement outputs

- the string literal `" and the area is "`, followed by
- the value of the `REAL` variable named `area`, followed by
- the string literal `"."`

These items appear in sequence on the same line of output text.

On the other hand, look at the last **two** `PRINT` statements. Notice that they print their outputs onto two consecutive lines of output text, in the same sequence as they occur in the program.

Specifying the Value of a Variable Via List-Directed Input from the Keyboard

```
% cat read_variable.f90
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!! Program:  read_variable                     !!!
!!! Author:   Henry Neeman (hneeman@ou.edu)     !!!
!!! Course:   CS 1313 010 Spring 2003           !!!
!!! Lab:      Sec 012 Fridays 1:30pm            !!!
!!! Description:  Declares, inputs and then     !!!
!!!   outputs a variable.                       !!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
PROGRAM read_variable
!
! ************************************************
! * Declaration section                          *
! ************************************************
!
! * All variables must be explicitly declared.
!
    IMPLICIT NONE
!
! *******************
! * Local variables *
! *******************
!
! * height_in_cm:  my height in cm
!
    INTEGER :: height_in_cm
!
! ************************************************
! * Execution section                            *
! ************************************************
!
! * Prompt the user to input the value of
! * height_in_cm.
!
    PRINT *, "What’s my height in centimeters?"
!
! * Input height_in_cm from standard input (i.e.,
! * the keyboard).
!
    READ *, height_in_cm
!
! * Print height_in_cm to standard output.
!
    PRINT *, "My height is ", height_in_cm, " cm."
END PROGRAM read_variable
% f95 -o read_variable read_variable.f90
% read_variable
 What’s my height in centimeters?
 160
 My height is  160  cm.
```


Multiple Variables Per READ Statement

Fortran 90 supports inputting multiple variables per READ statement.

The individual input values can be separated

- by commas,
- by blank spaces, and/or
- by carriage returns.

```
% cat readlist.f90
PROGRAM readlist
    IMPLICIT NONE
    INTEGER :: number_of_silly_people, number_of_toys
    REAL    :: average_height_in_m
    LOGICAL :: toys_put_away
    CHARACTER (LEN = 20) :: username, hometown

    PRINT *, "How many silly people are there in CS1313,"
    PRINT *, "  and what’s their average height in meters?"
    READ *, number_of_silly_people, average_height_in_m
    PRINT *, "There are ", number_of_silly_people, &
 &           " silly people"
    PRINT *, " with an average height of ", &
 &           average_height_in_m, " m."
    PRINT *, "How many toys do I have, and is it true"
    PRINT *, "  that I put them away?"
    READ *, number_of_toys, toys_put_away
    PRINT *, "I have ", number_of_toys, " toys."
    PRINT *, "My toys are put away? : ", toys_put_away
    IF (toys_put_away) THEN
        PRINT *, "  Yes, I did put my toys away."
    ELSE
        PRINT *, "  No, I didn’t put my toys away."
    END IF
    PRINT *, "What’s my username and hometown?"
    READ *, username, hometown
    PRINT *, "My username is ", username
    PRINT *, "and my hometown is ", hometown, "."
END PROGRAM readlist
% f95 -o readlist readlist.f90
% readlist
 How many silly people are there in CS1313,
   and what’s their average height in meters?
 7,  1.75
 There are  7  silly people
  with an average height of    1.7500000  m.
 How many toys do I have, and is it true
   that I put them away?
 43  T
 I have  43  toys.
 My toys are put away? :  T
   Yes, I did put my toys away.
 What’s my username and hometown?
 neeman
    Buffalo
 My username is neeman
 and my hometown is Buffalo            .
```

Program Variables vs. Algebra Variables

Variables in Fortran 90 (and many other programming languages) look and feel very similar to variables that you deal with in your math classes, from high school algebra on up.

This is on purpose.

The main difference between an algebra variable and a program variable is that a program variable can change its value during a run:

Algebra	**Fortran 90**	**Output**
Let x be 5.	`x = 5`	
$\therefore x = 5$	`PRINT *, x`	`5`
Let y be 7.	`y = 7`	
$\therefore y = 7$	`PRINT *, y`	`7`
$z = x + y$	`z = x + y`	
$\therefore z = 12$	`PRINT *, z`	`12`
	`z = x * y`	
	`PRINT *, z`	`35`

Programming Exercise

Create a program that:

1. prompts the user for their weight in pounds;
2. inputs the user's weight in pounds;
3. outputs the user's weight in pounds.

Begin by drawing a flowchart, and then write the program. The program **MUST** have an `IMPLICIT NONE` statement, but does not have to have comments. The data type for the weight variable must be appropriate.