CIS1400 – Programming Logic and Technique

Topic 3 → Understanding Modules

Chapter Topics

3.1 Introduction

3.2 Defining and Calling a Module

3.3 Local Variables

3.4 Passing Arguments to Modules

3.5 Global Variables and Global Constants

Sample Program: Fluid Conversion

3.1 Introduction

- A module is a group of statements that exists within a program for the purpose of performing a specific task.
- Most programs are large enough to be broken down into several subtasks.
- Divide and conquer: It's easier to tackle smaller tasks individually.

3.1 Introduction

5 benefits of using modules

- Simpler code
  - Small modules easier to read than one large one
- Code reuse
  - Can call modules many times
- Better testing
  - Test separate and isolate then fix errors
- Faster development
  - Reuse common tasks
- Easier facilitation of teamwork
  - Share the workload

3.2 Defining and Calling a Module

- Like variables, programmers define module names following certain rules
  - Should be descriptive enough so that anyone reading the code can guess what the module does.
  - No spaces in a module name.
  - No punctuation.
  - Cannot begin with a number.

3.2 Defining and Calling a Module

- The code for a module is known as a module definition.
- Definition contains two parts
  - A header
    - The starting point of the module. Specifies name of module and any parameter values
  - A body
    - The statements that execute when the module is called.
  - Pseudocode Format:

```
Module NameOfModule(ParameterList)
        Statement
        Statement
        Etc.
End Module
```

header

body

3.2 Defining and Calling a Module

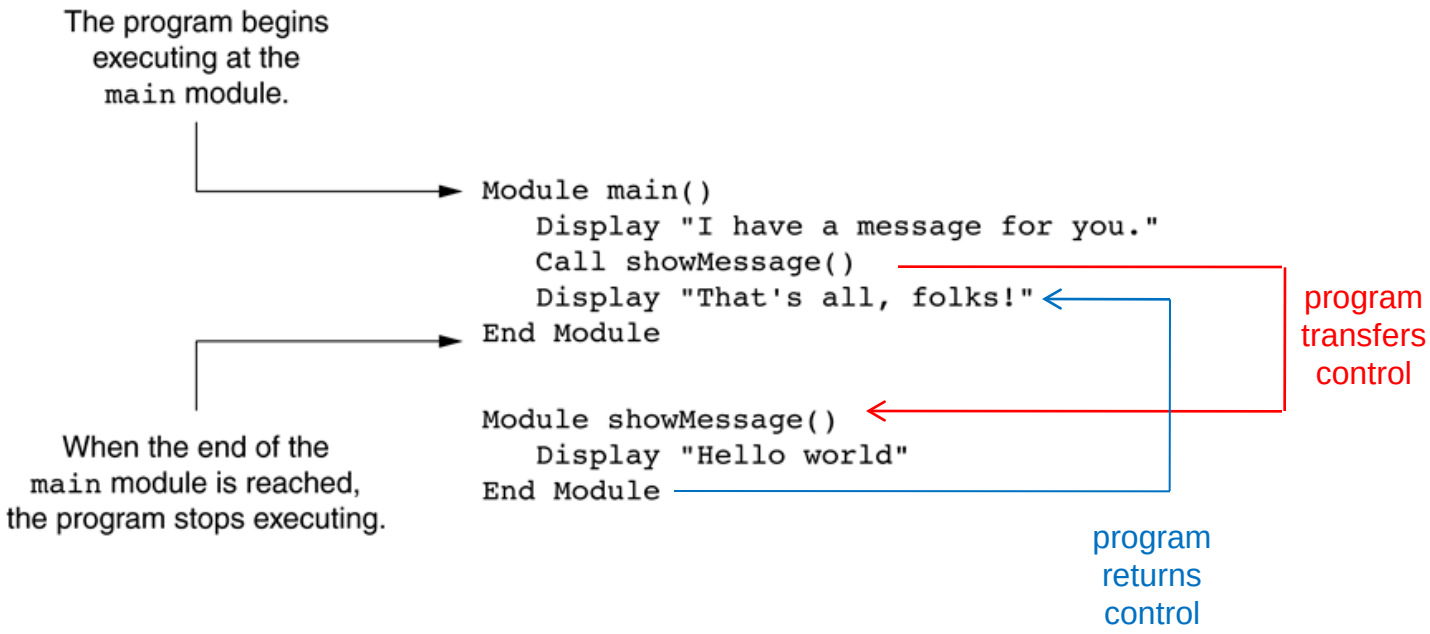
- A call must be made to the module in order for the statements in the body to execute.
  - Pseudocode Format:

```
Module main( )
        Statement
         Call NameOfModule(ArgumentList)
        Statement
        Etc.
End Module
```

Call statement

3.2 Defining and Calling a Module

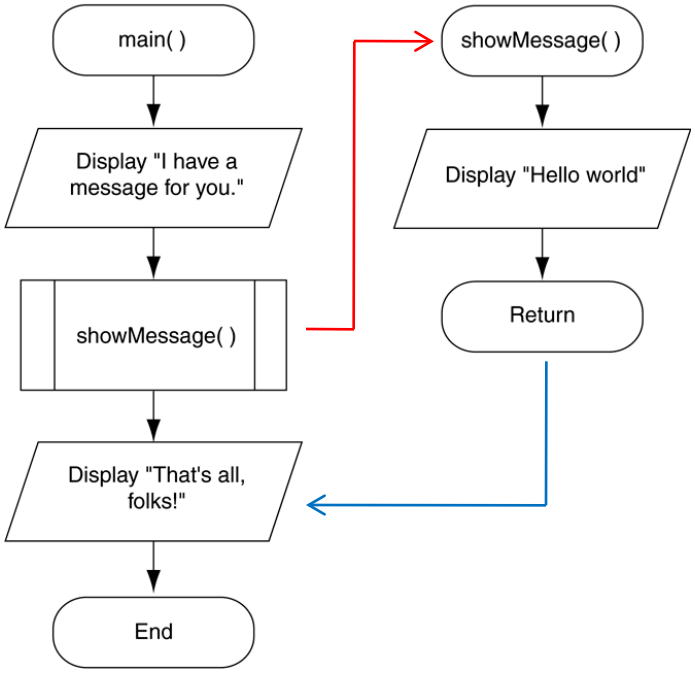
Figure 3-2, 3-3, 3-4 The `main` and `showMessage` modules



3.2 Defining and Calling a Module

- When flowcharting a program with modules, each module is drawn separately.

Figure 3-6 Flowchart for Program 3-1



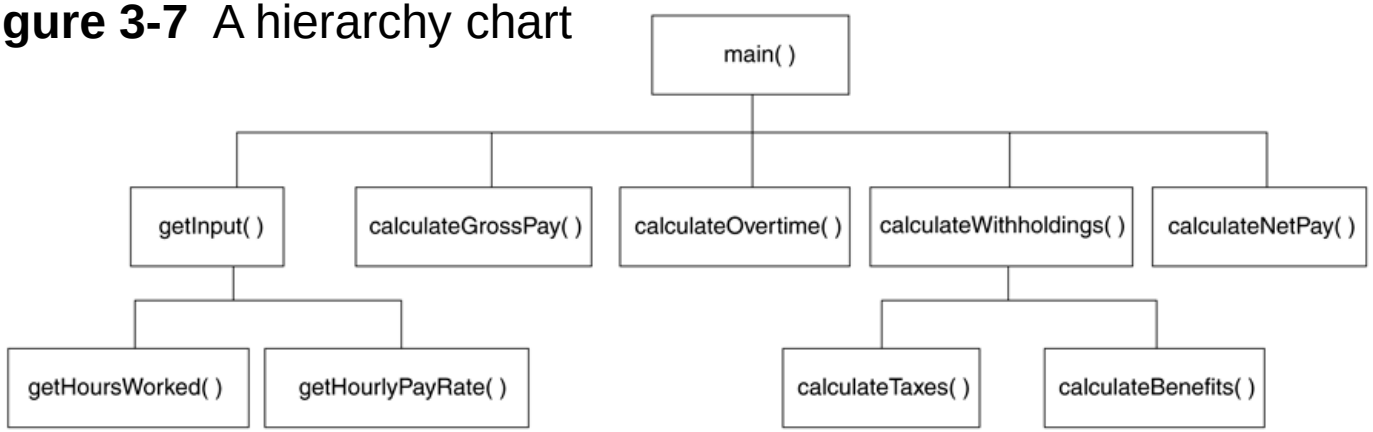
3.2 Defining and Calling a Module

- A top-down design is used to break down an algorithm into modules by the following steps:
  - The overall task is broken down into a series of subtasks.
  - Each of the subtasks is repeatedly examined to determine if it can be further broken down.
  - Each subtask is coded.

3.2 Defining and Calling a Module

- A hierarchy chart gives a visual representation of the relationship between modules.
  - Also called structure chart.
- The details of the program are excluded.
  - Flowchart/Pseudocode used for module details.

Figure 3-7 A hierarchy chart



3.3 Local Variables

- A **local variable** is declared inside a module and cannot be accessed by statements that are outside the module.

```
Module main()
  Call getName()
  Display "Hello ", name
End Module

Module getName()
  Declare String name
  Display "Enter your name."
  Input name
End Module
```

This will cause an error! The variable name is not visible in this module.

The local variable name is only visible in the module 'getName'.

3.3 Local Variables

- A **local variable** is declared inside a module and cannot be accessed by statements that are outside the module.
- **Scope** describes the part of the program in which a variable can be accessed.
- Variables with the **same** scope must have **different** names.
  - Error to redefine a variable in same scope!

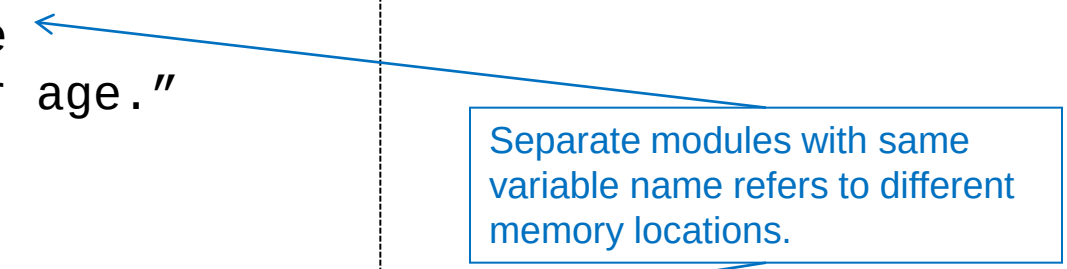
```
Module getTwoAges()
  Declare Integer age
  Display "Enter your age."
  Input age

  Declare Integer age
  Display "Enter your pet's age."
  Input age
End Module
```

This will cause an error!
A variable named age has already been declared in the same module.

3.3 Local Variables

- Variables in **different** scope can have the **same** names.
  - They are **different** variables in **different** memory locations.

```
Module getYourAge()
  Declare Integer age
  Display "Enter your age."
  Input age
End Module

Module getPetsAge()
  Declare Integer age
  Display "Enter your pet's age."
  Input age
End Module
```

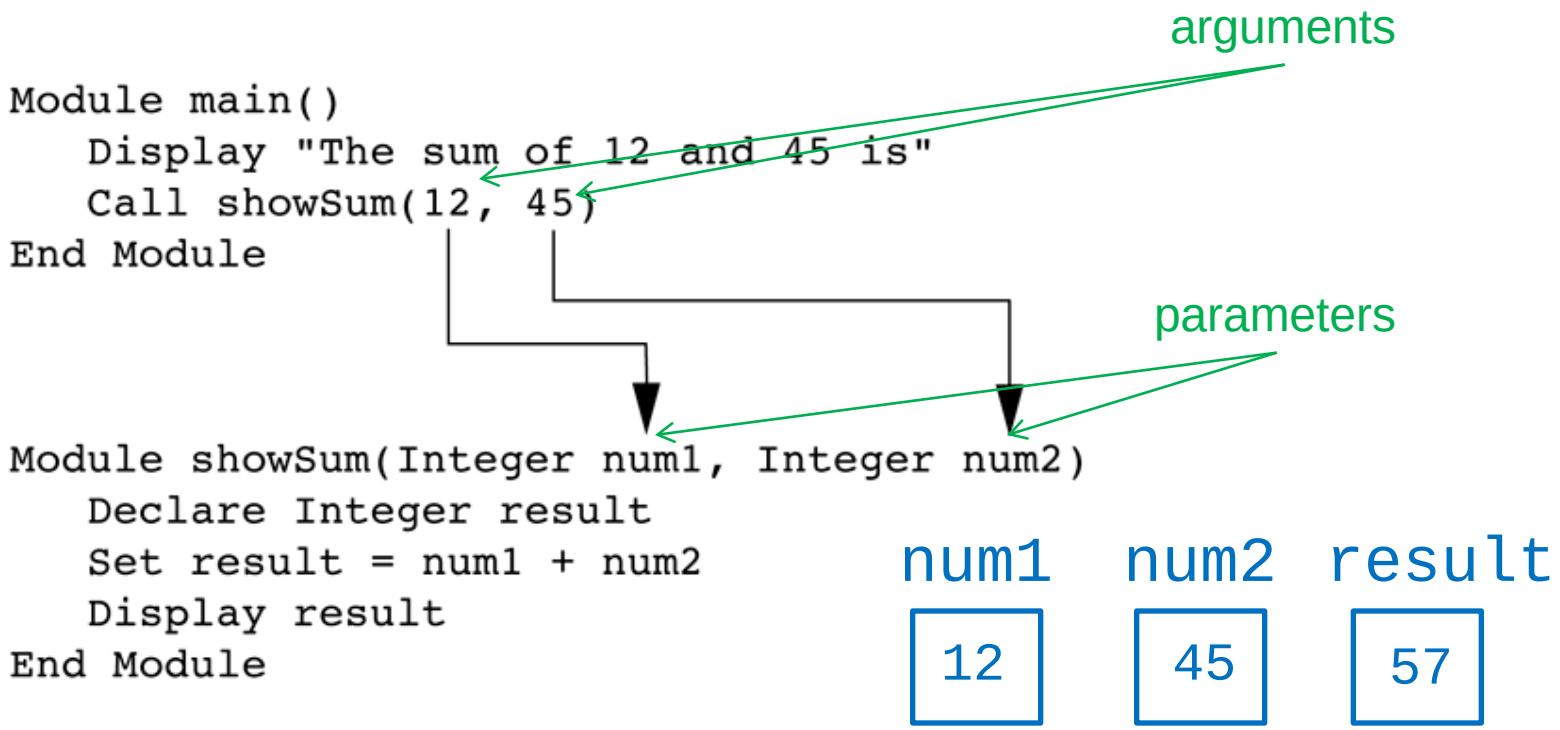


3.4 Passing Arguments to Modules

- Sometimes, one or more pieces of data need to be **shared** among modules.
- An **argument** is any piece of data that is passed into a module when the module is called.
- A **parameter** is a variable that receives an argument that is passed into a module.
  - Parameter includes data type and parameter name
  - Treated as initialized local variable
- The **argument** and the receiving **parameter** variable must be of the **same data type**.
- Multiple arguments can be passed sequentially into a **parameter list**.
  - They are separated by commas.

3.4 Passing Arguments to Modules

Figure 3-14 Two arguments passed (by value) into two parameters



3.4 Passing Arguments to Modules

Pass by Value vs. Pass by Reference

- Pass by **Value** means that only a **copy** of the argument's value is passed into the module.
  - One-directional communication:
    - Calling module **can** only communicate with the called module
    - Called module **cannot** modify the value of the argument.
- Pass by **Reference** means that the argument is passed into a **reference** variable.
  - Two-way communication:
    - Calling module **can** communicate with called module;
    - Called module **can** modify the value of the argument.
  - Use keyword "Ref" in module header

3.4 Passing Arguments to Modules

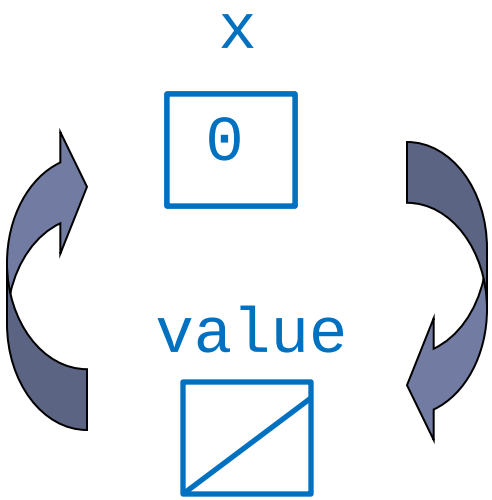
Reference variable acts as an alias for the passed variable

Should only pass variable arguments by reference

```
Module main()
  Declare Integer x = 99
  Display “x is set to “, x
  Call setToZero(x)
  Display “x is set to “, x
End Module

Module setToZero(Integer Ref value)
  Set value = 0
End Module
```

Program Output
x is set to 99
x is set to 0

Pass by reference keyword



3.5 Global Variables & Global Constants

- Global variable and constants defined outside modules, at top of program
- A **global variable** is accessible to all modules.
- Should be avoided because:
  - They make debugging difficult
  - Making the module dependent on global variables makes it hard to reuse module in other programs
  - They make a program hard to understand

```
Declare Integer number

Module main()
   Display "Enter a number."
   Input number
   Call showNumber()
End Module

Module showNumber()
   Display "The number you entered is ", number
End Module
```

Global variable declared and accessed from separate modules.

3.5 Global Variables & Global Constants

- Global variable and constants defined outside modules, at top of program
- A **global constant** is a named constant that is available to every module in the program.
  - Since a program cannot modify the value of a constant, these are *safer* than global variables.

```
Constant Real CONTRIBUTION_RATE = 0.95

Module showGrossPayContrib(Real grossPay)
   Declare Real contrib
   Set contrib = grossPay * CONTRIBUTION_RATE
   Display "The contribution for the gross pay"
   Display "is $", contrib
End Module
```

Global constant declared and accessed.

Sample Program: Fluid Conversion

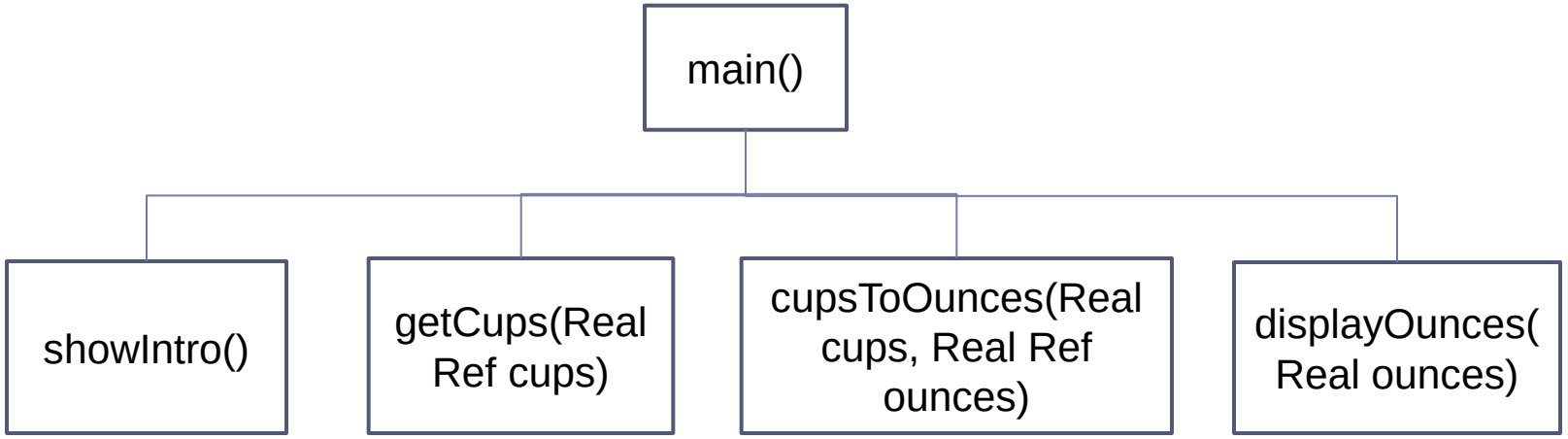
Modification of "*In The Spotlight*" page 104

- Convert cups into fluid ounces
  - 1 Cup = 8 Fluid Ounces
  - Ounces = Cups * 8
- ***What is required for each phase of the program?***

1. What must be read as input?
   - Get number of cups
2. How will the input be processed?
   - Multiply the number of cups by the ounces per cup
3. What will be done with the output?
   - Display the number of ounces

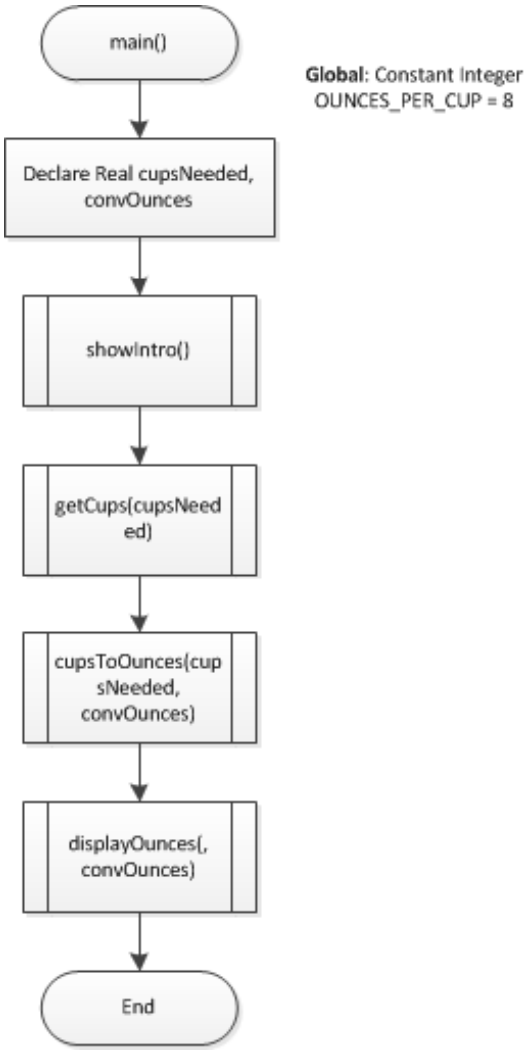
Sample Program: Fluid Conversion

- "*When using modules in a program, you generally isolate each task within the program in its own module.*" (p80)
  - Display program purpose
  - Get input
  - Process input
  - Display output



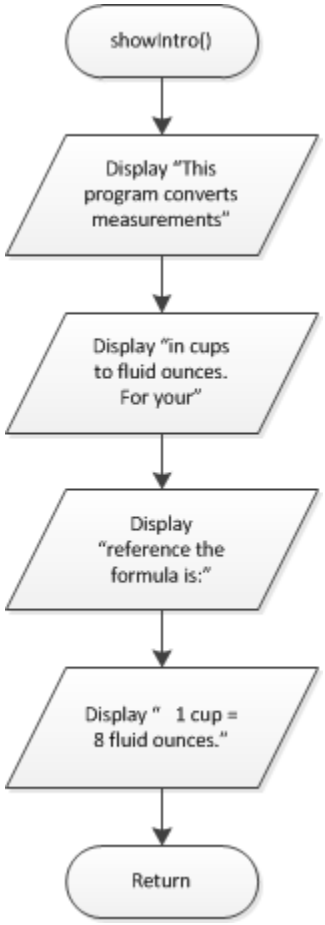
Sample Program: Fluid Conversion

```
// Global Constant
Constant Integer OUNCES_PER_CUP = 8

Module main()
  // Declare needed variables
  Declare Real cupsNeeded, convOunces

  // Display intro message
  Call showIntro()

  // Get the number of cups
  Call getCups(cupsNeeded)

  // Convert cups to ounces
  Call cupsToOunces(cupsNeeded, convOunces)

  // Display converted amount
  Call displayOunces(convOunces)
End Module
```



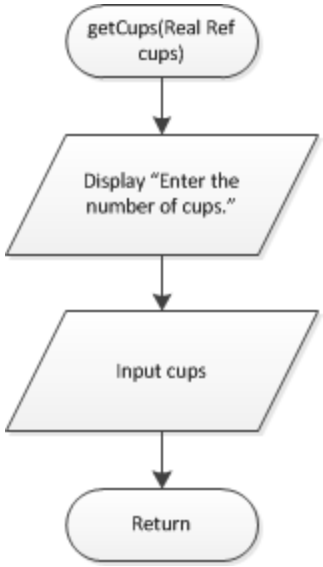
Sample Program: Fluid Conversion

```
// The showIntro module display an
// introductory screen
Module showIntro()
  Display “This program converts measurements”
  Display “in cups to fluid ounces.  For your”
  Display “reference the formula is:”
  Display “   1 cup = 8 fluid ounces.”
End Module
```



Sample Program: Fluid Conversion

```
// The getCups module gets the number of cups
// and stores it in the reference variable cups.
Module getCups(Real Ref cups)
  Display “Enter the number of cups.”
  Input cups
End Module
```



Sample Program: Fluid Conversion

```
// The cupsToOunces module takes an input number
// of cups and returns the converted ounces
// in the passed reference variable.
Module cupsToOunces(Real cups, Real Ref ounces)
  Set ounces = cups * OUNCES_PER_CUP
End Module
```

```
// The displayOunces module displays the
// equivalent number of ounces.
Module displayOunces(Real ounces)
  Display “That converts to “, ounces, “ ounces.”
End Module
```