Programming Logic & Design

Chapter 3 – Modules

Sagara Samarawickrama

CSD 1133 – CSAM

Chapter 3 – Modules

Topics:

- Introduction to Modules
- Defining and Calling a Module
- Local Variables
- Passing Arguments to Modules
- Global Variables and Global Constants

Chapter 3 – Modules

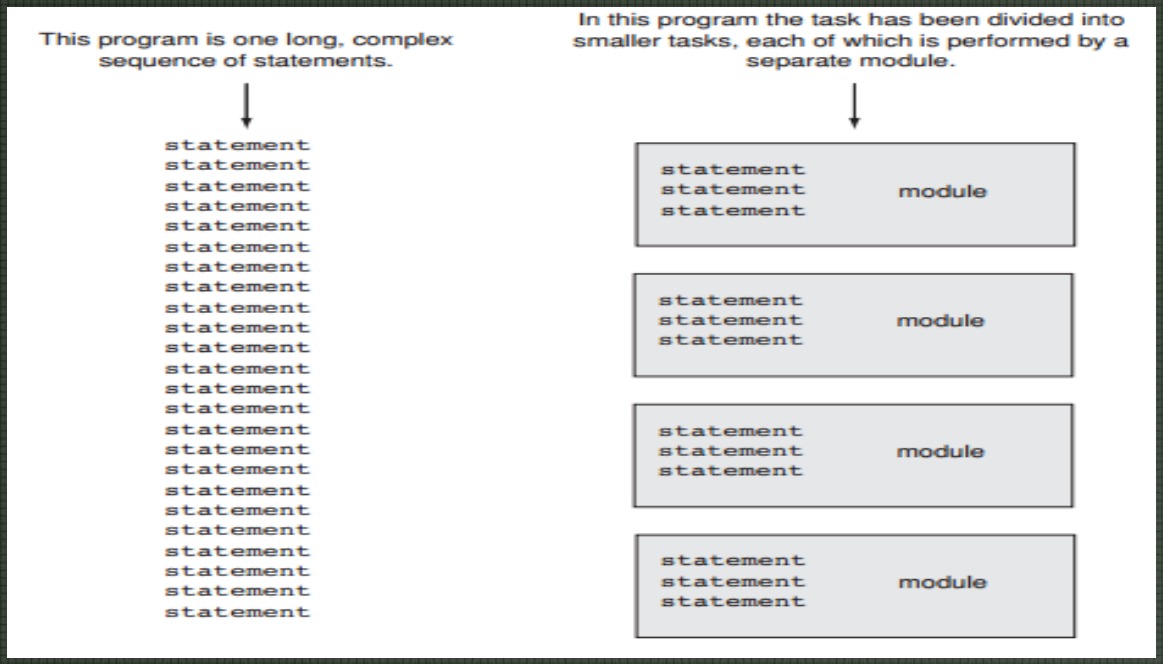
Introduction to Modules

Most programs perform tasks that are large enough to be broken down into several subtasks. For this reason, programmers usually break down their programs into modules. A module is a group of statements that exist within a program for the purpose of performing a specific task. Instead of writing a large program as one long sequence of statements, it can be written as several small modules, each one performing a specific part of the task. These small modules can then be executed in the desired order to perform the overall task.

This approach is sometimes called divide and conquer because a large task is divided into several smaller tasks that are easily performed. Figure in the next page illustrates this idea by comparing two programs: one that uses a long, complex sequence of statements to perform a task, and another that divides a task into smaller tasks, each of which are performed by a separate module.

Although every modern programming language allows you to create modules, they are not always referred to as modules. Modules are commonly called procedures, subroutines, subprograms, methods, and functions.

Chapter 3 – Modules



Benefits of using module:

- **Simpler Code**
- **Code Reuse**
- **Better Testing**
- **Faster development**
- **Easier Facilitation of Teamwork**

Chapter 3 – Modules

Checkpoint :

What is a module?
What is meant by the phrase “divide and conquer”?
How do modules help you reuse code in a program?

Defining and Calling a Module

Because modules perform actions, most programmers prefer to use verbs in module names. For example, a module that calculates gross pay might be named *calculateGrossPay.* When naming a module, most languages require that you follow the same rules that you follow when naming variables. This means that module names cannot contain spaces, cannot typically contain punctuation characters, and usually cannot begin with a number. These are only general rules, however. The specific rules for naming a module will vary slightly with each programming language.

To create a module you write its definition. In most languages, a module definition has two parts: a header and a body. The header indicates the starting point of the module, and the body is a list of statements that belong to the module.

Chapter 3 – Modules

```
Module name()
   statement
   statement
   etc.
End Module
```

These statements are the body of the module.

A module definition specifies what a module does, but it does not cause the module to execute. To execute a module, we must call it. In pseudocode we will use the word Call to call a module.

This is how we would call the *showMessage* module:

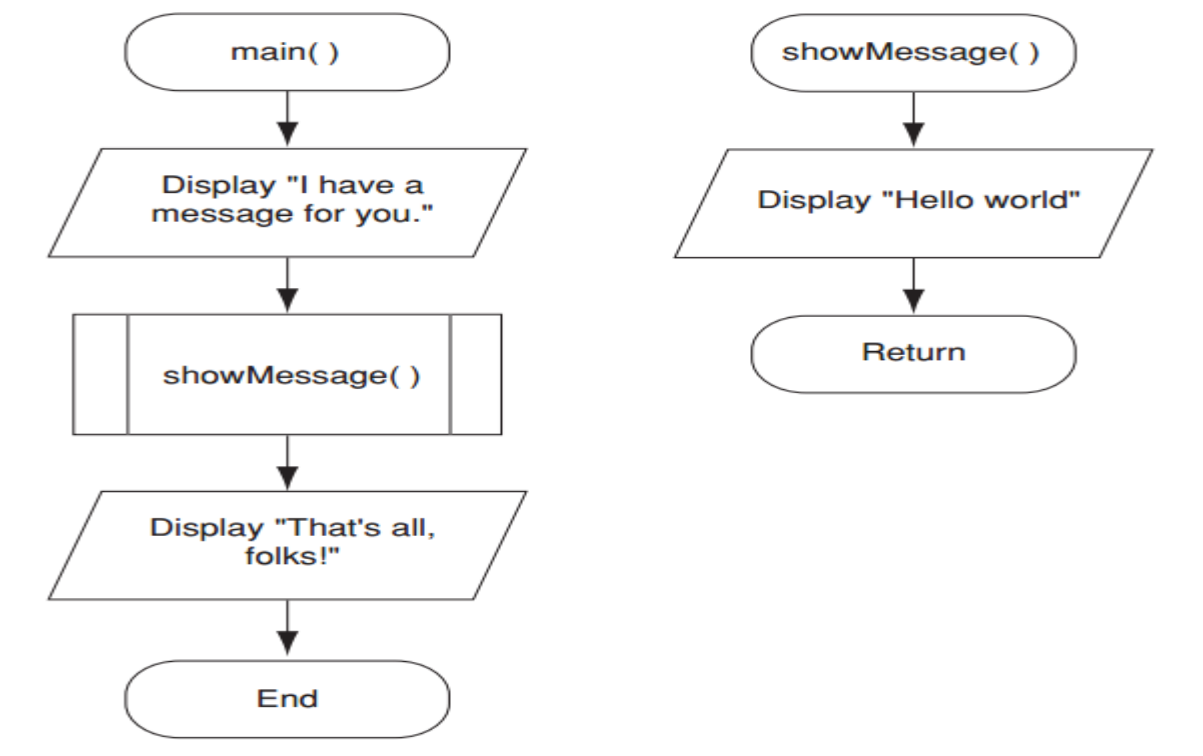
```
Call showMessage()
```

```
1 Module main()
2    Display "I have a message for you."
3    Call showMessage()
4    Display "That's all, folks!"
5 End Module
```



```
7 Module showMessage()
8    Display "Hello world"
9 End Module
```

Chapter 3 – Modules



Just as important as understanding how modules work is understanding how to design a modularized program. Programmers commonly use a technique known as top-down design to break down an algorithm into modules. The process of top-down design is performed in the following manner:

Chapter 3 – Modules

● The overall task that the program is to perform is broken down into a series of subtasks.
● Each of the subtasks is examined to determine whether it can be further broken down into more subtasks. This step is repeated until no more subtasks can be identified.
● Once all of the subtasks have been identified, they are written in code.

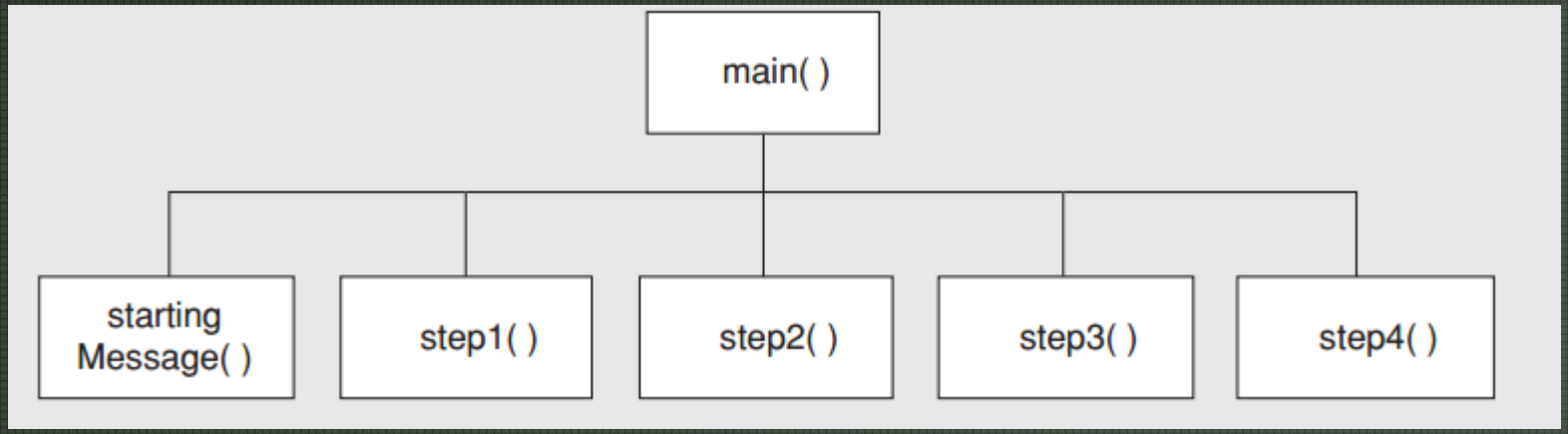
Problem :

Professional Appliance Service, Inc. offers maintenance and repair services for household appliances. The owner wants to give each of the company's service technicians a small handheld computer that displays step-by-step instructions for many of the repairs that they perform. To see how this might work, the owner has asked you to develop a program that displays the following instructions for disassembling an ACME laundry dyer:
Step 1: Unplug the dryer and move it away from the wall.
Step 2: Remove the six screws from the back of the dryer.
Step 3: Remove the dryer's back panel.
Step 4: Pull the top of the dryer straight up.

Chapter 3 – Modules

Professional Appliance Service, Inc. offers maintenance and repair services for household appliances. The owner wants to give each of the company's service technicians a small handheld computer that displays step-by-step instructions for many of the repairs that they perform. To see how this might work, the owner has asked you to develop a program that displays the following instructions for disassembling an ACME laundry dyer:

Step 1: Unplug the dryer and move it away from the wall.
Step 2: Remove the six screws from the back of the dryer.
Step 3: Remove the dryer's back panel.
Step 4: Pull the top of the dryer straight up.



Chapter 3 – Modules

Solution :

```
Module main()

// Display the starting message.
 Call startingMessage()
 Display "Press a key to see Step 1."
 Input
 // Display Step 1.
 Call step1()
 Display "Press a key to see Step 2."
 Input
// Display Step 2.
 Call step2()
 Display "Press a key to see Step 3."
 Input
// Display Step 3.
 Call step3()
 Display "Press a key to see Step 4."
 Input
// Display Step 4.
 Call step4()
 End Module
```

Chapter 3 – Modules

Solution :

```
Module main()

// Display the starting message.
Call startingMessage()
Display "Press a key to see Step 1."
 // The startingMessage module displays  the program's starting message.
Module startingMessage()
Display "This program tells you how to"
Display "disassemble an ACME laundry dryer."
Display "There are 4 steps in the process."
End Module

// The step1 module displays the instructions  for Step 1.
Module step1()
Display "Step 1: Unplug the dryer and"
Display "move it away from the wall."
End Module
// The step2 module displays the instructions / for Step 2.
Module step2()
Display "Step 2: Remove the six screws"
Display "from the back of the dryer."
End Module
```

Chapter 3 – Modules

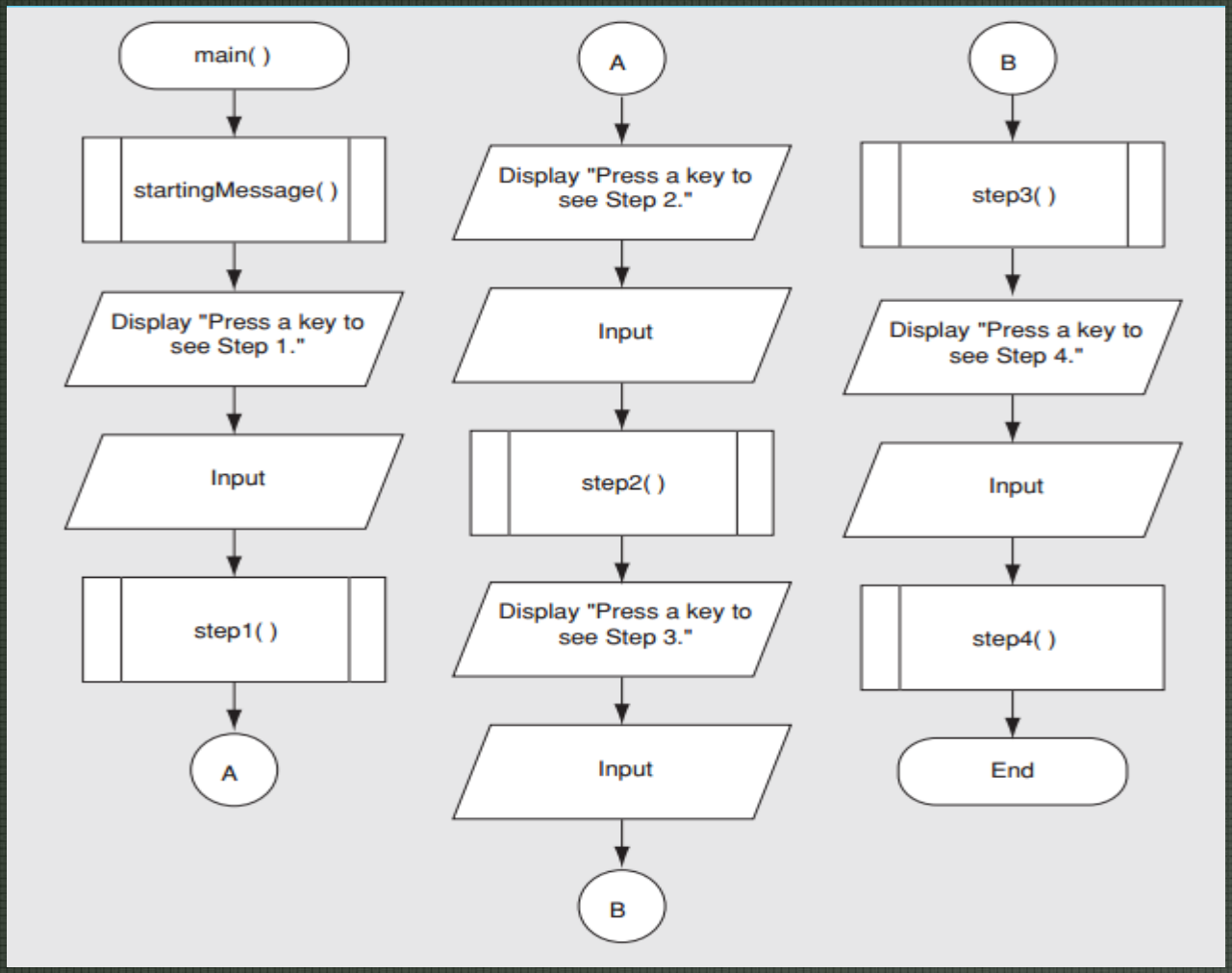
```
// The step3 module displays the instructions for Step 3.
Module step3()
Display "Step 3: Remove the dryer's"
Display "back panel."
End Module
// The step4 module displays the instructions  for Step 4.
Module step4()
Display "Step 4: Pull the top of the"
Display "dryer straight up."
End Module
```

Program Output

```
This program tells you how to
disassemble an ACME laundry dryer.
There are 4 steps in the process.
Press a key to see Step 1.
[Enter]
Step 1: Unplug the dryer and
move it away from the wall.
Press a key to see Step 2.
[Enter]
Step 2: Remove the six screws
from the back of the dryer.
Press a key to see Step 3.
[Enter]
Step 3: Remove the dryer's
back panel.
Press a key to see Step 4.
[Enter]
Step 4: Pull the top of the
dryer straight up.
```

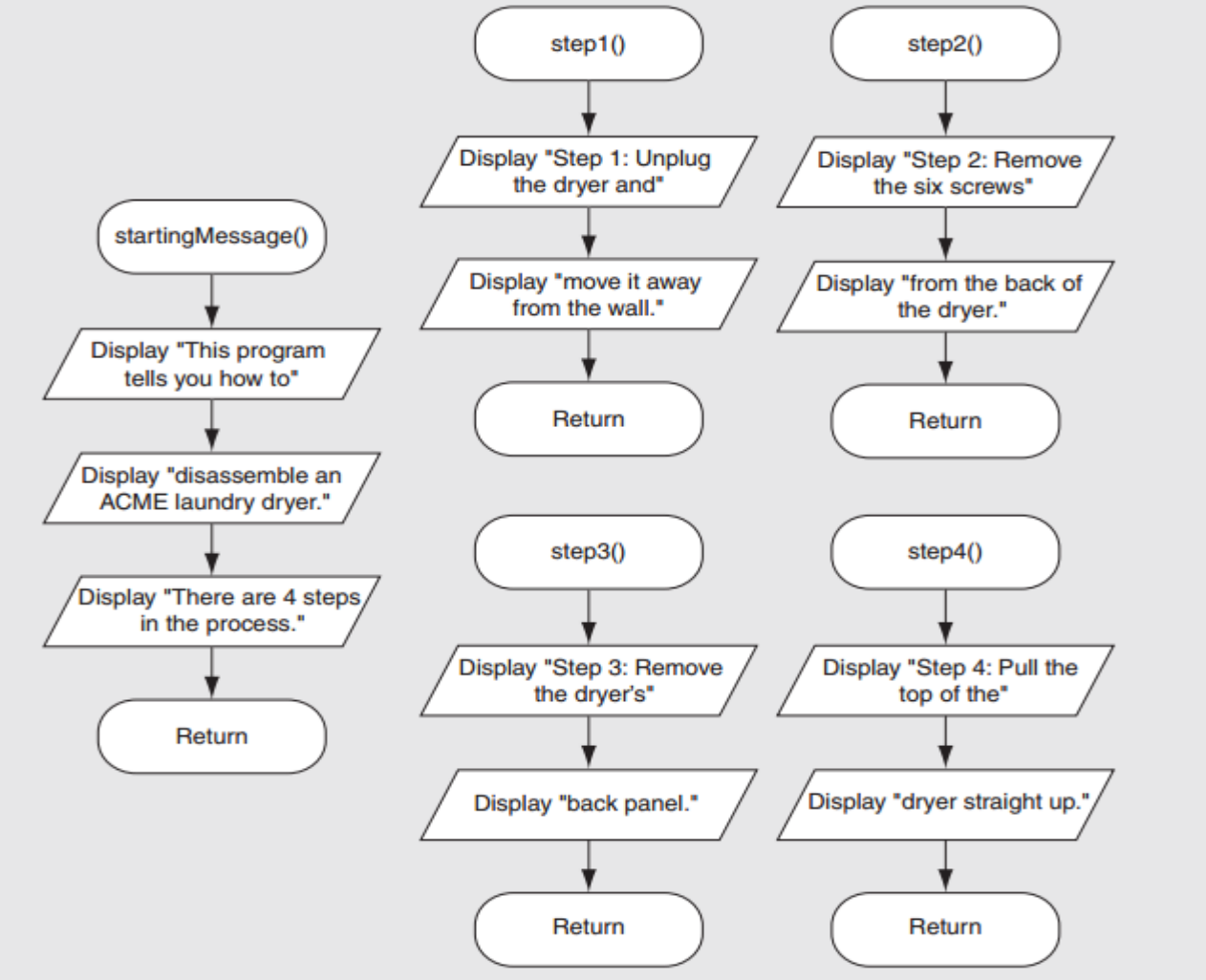
Chapter 3 – Modules

Flowchart



Chapter 3 – Modules

Flowchart …



Chapter 3 – Modules

Local Variables

In most programming languages, a variable that is declared inside a module is called a local variable. A local variable belongs to the module in which it is declared and only statements inside that module can access the variable.

```
1  Module main()
2     Call getName()
3     Display "Hello ", name      <-- This will cause an error!
4  End Module
5
6  Module getName()
7     Declare String name         <-- This is local variable.
8     Display "Enter your name."
9     Input name
10 End Module
```

A local variable's scope usually begins at the variable's declaration and ends at the end of the module in which the variable is declared. The variable cannot be accessed by statements that are outside this region. This means that a local variable cannot be accessed by code that is outside the module, or inside the module but before the variable's declaration.

```
Module getName()
   Display "Enter your name."
   Input name          <-- This statement will cause an error because
   Declare String name        the name variable has not been declared yet.
End Module
```

Chapter 3 – Modules

In most programming languages, you cannot have two variables with the same name in the same scope. For example, look at the following module:

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

   Declare Integer age      <-- This will cause an error!
   Display "Enter your pet's age."   A variable named age has
   Input age                         already been declared.
End Module
```

Checkpoint

1. What is a local variable? How is access to a local variable restricted?
2. What is a variable's scope?
3. Is it usually permissible to have more than one variable with the same name in the same scope? Why or why not?
4. Is it usually permissible for a local variable in one module to have the same name as a local variable in a different module?

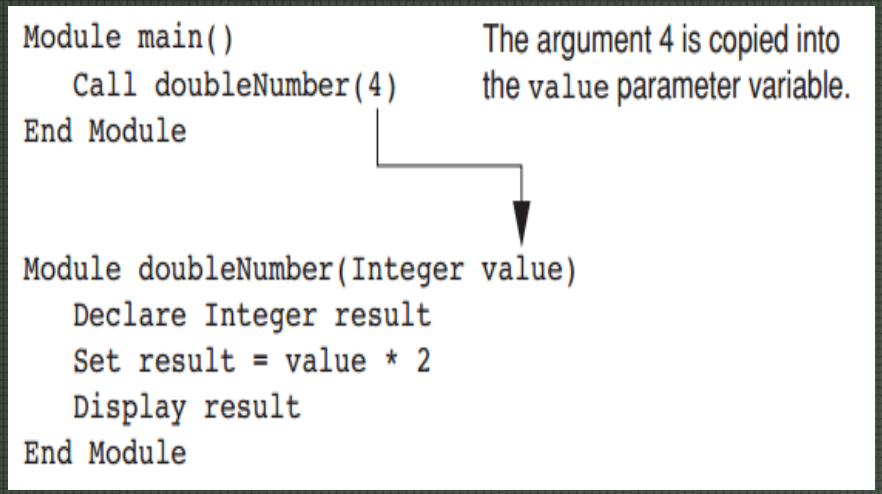
Chapter 3 – Modules

Passing Arguments to Modules

It is useful not only to call a module, but also to send one or more pieces of data into the module. Pieces of data that are sent into a module are known as arguments.

```
Module main()
Call doubleNumber(4)
End Module

Module doubleNumber(Integer value)
Declare Integer result
Set result = value * 2
Display result
End Module
```



When you pass an argument to a module, most programming languages require that the argument and the receiving parameter variable be of the same data type. If you try to pass an argument of one type into a parameter variable of another type, an error usually occurs.

Chapter 3 – Modules

```
    Call doubleNumber(number)
End Module

Module doubleNumber(Integer value)
    Declare Integer result
    Set result = value * 2
    Display result
End Module
```

20

```
    Call doubleNumber(55.9)
End Module

Module doubleNumber(Integer value)
    Declare Integer result
    Set result = value * 2
    Display result
End Module
```

Error!

A parameter variable's scope is usually the entire module in which the parameter is declared. No statement outside the module can access the parameter variable. Most languages allow you to write modules that accept multiple arguments. Following shows a pseudocode module named showSum, that accepts two Integer arguments.

```
Module main()
    Display "The sum of 12 and 45 is:"
    Call showSum(12, 45)
End Module

Module showSum(Integer num1, Integer num2)
    Declare Integer result
    Set result = num1 + num2
    Display result
End Module
```

Chapter 3 – Modules

Problem :

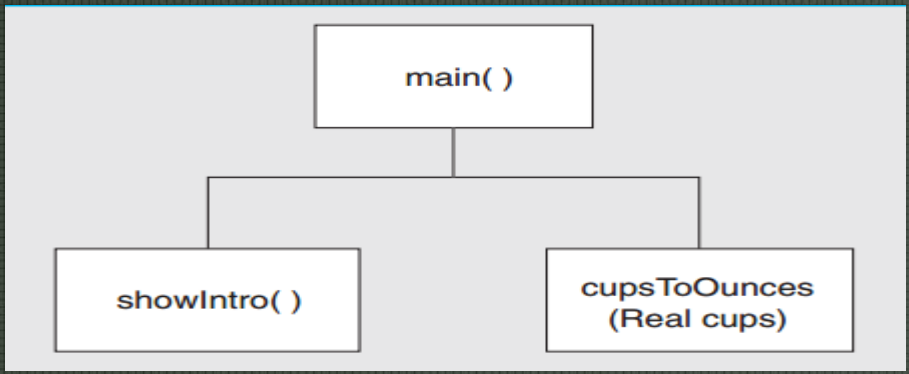
Your friend Michael runs a catering company. Some of the ingredients that his recipes require are measured in cups. When he goes to the grocery store to buy those ingredients, however, they are sold only by the fluid ounce. He has asked you to write a simple program that converts cups to fluid ounces.

You design the following algorithm:

1. Display an introductory screen that explains what the program does.
2. Get the number of cups.
3. Convert the number of cups to fluid ounces and display the result.

Solution :

The program's structure in a hierarchy chart is as follows



Chapter 3 – Modules

Pseudocode for the program

```
Module main()
// Declare a variable for the  number of cups needed.
Declare Real cupsNeeded

// Display an intro message.
Call showIntro()

// Get the number of cups.
Display "Enter the number of cups."
Input cupsNeeded

// Convert cups to ounces.
Call cupsToOunces(cupsNeeded)
End Module
```

Chapter 3 – Modules

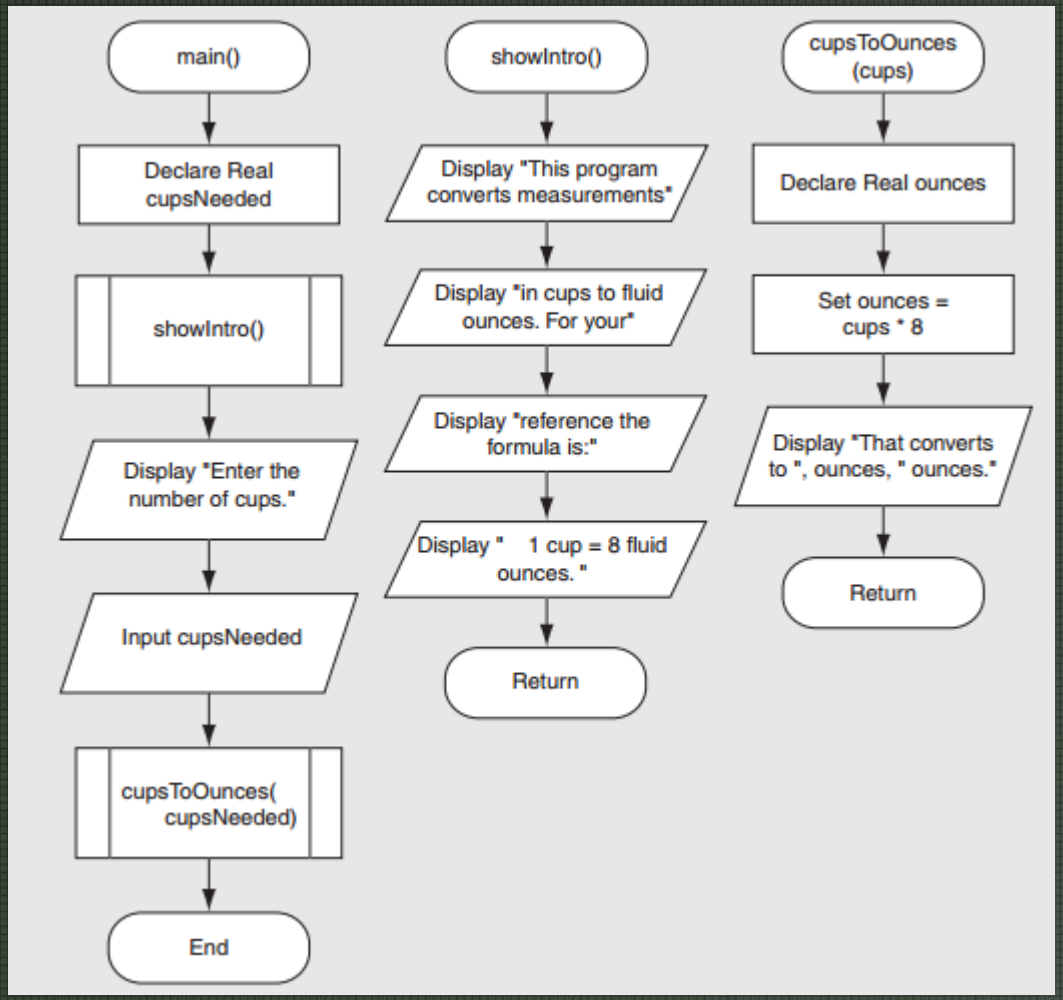
Pseudocode for the program

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

// The cupsToOunces module accepts a number of cups and displays the equivalent number of ounces.
Module cupsToOunces(Real cups)
// Declare variables.
Declare Real ounces
// Convert cups to ounces.
Set ounces = cups * 8
// Display the result.
Display "That converts to ",ounces, " ounces."
End Module
```

Chapter 3 – Modules

Flowchart:



Chapter 3 – Modules

Passing Arguments by Value and by Reference

Many programming languages provide two different ways to pass arguments: by value and by reference. Before studying these techniques in detail, we should mention that different languages have their own way of doing each.

Passing Arguments by Value

Passing an argument by *value* means that only a copy of the argument's value is passed into the parameter variable. If the contents of the parameter variable are changed inside the module, it has no effect on the argument in the calling part of the program.

```
Module main()
Declare Integer number = 99
// Display the value stored in number.
Display "The number is ", number
// Call the changeMe module, passing  the number variable as an argument.
 Call changeMe(number)
 // Display the value of number again.
Display "The number is", number
End Module
```

Chapter 3 – Modules

```
Module changeMe(Integer myValue)
Display "I am changing the value."
// Set the myValue parameter variable  to 0.
Set myValue = 0
// Display the value in myValue.
Display "Now the number is ", myValue
End Module
```

Program Output

```
The number is 99
I am changing the value.
Now the number is 0
The number is 99
```

Passing Arguments by Reference

Passing an argument by reference means that the argument is passed into a special type of parameter known as a reference variable. When a reference variable is used as a parameter in a module, it allows the module to modify the argument in the calling part of the program.

Chapter 3 – Modules

```
Module main()
// Declare and initialize some variables.
Declare Integer x = 99
Declare Integer y = 100
Declare Integer z = 101

// Display the values in those variables.
Display "x is set to ", x
Display "y is set to ", y
Display "z is set to ", z

// Pass each variable to setToZero.
Call setToZero(x)
Call setToZero(y)
Call setToZero(z)

// Display the values now.
Display "---------------"
Display "x is set to ", x
Display "y is set to ", y
Display "z is set to ", z
End Module

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

Program Output

```
x is set to 99
y is set to 100
z is set to 101
---------------
x is set to 0
y is set to 0
z is set to 0
```

Chapter 3 – Modules

Programming Task:

In the previous programming task, we developed a program that your friend Michael can use in his catering business. The program does exactly what Michael wants it to do: it converts cups to fluid ounces. After studying the program that we initially wrote, however, you believe that you can improve the design.

```
Module main()
   // Declare a variable for the
   // number of cups needed.
   Declare Real cupsNeeded

   // Display an intro message.
   Call showIntro()

   // Get the number of cups.
   Display "Enter the number of cups."
   Input cupsNeeded

   // Convert cups to ounces.
   Call cupsToOunces(cupsNeeded)
End Module
```

This code can be put in its own module.

Write a pseudocode and flowchart for the above program

Chapter 3 – Modules

Global Variables

A global variable is a variable that is visible to every module in the program. A global variable's scope is the entire program, so all of the modules in the program can access a global variable. In most programming languages, you create a global variable by writing its declaration statement outside of all the modules, usually at the top of the program

```
1  // The following declares a global Integer variable.
2  Declare Integer number
3
4  // The main module
5  Module main()
6     // Get a number from the user and store it
7     // in the global variable number.
8     Display "Enter a number."
9     Input number
10
11    // Call the showNumber module.
12    Call showNumber()
13 End Module
14
15 // The showNumber module displays the contents
16 // of the global variable number.
17 Module showNumber()
18    Display "The number you entered is ", number
19 End Module
```

Chapter 3 – Modules

Problem :

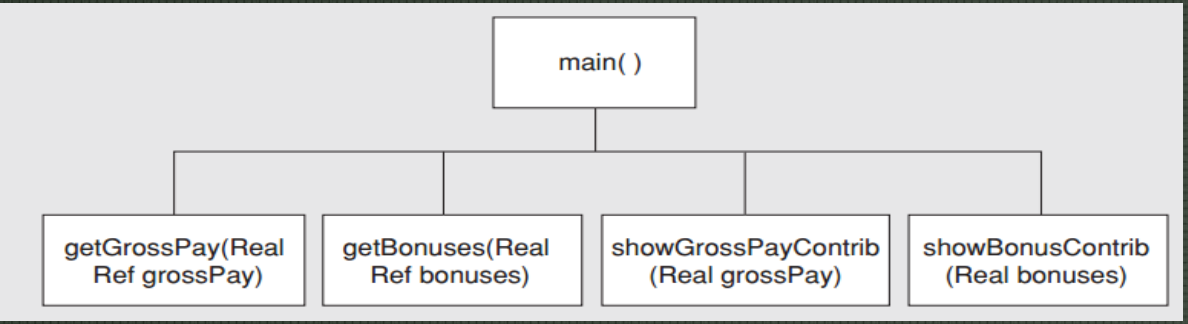
Marilyn works for Integrated Systems, Inc., a software company that has a reputation for providing excellent fringe benefits. One of its benefits is a quarterly bonus that is paid to all employees. Another benefit is a retirement plan for each employee. The company contributes 5 percent of each employee's gross pay and bonuses to his or her retirement plan. Marilyn wants to design a program that will calculate the company's contribution to an employee's retirement account for a year. She wants the program to show the amount of contribution for the employee's gross pay and for the bonuses separately

Solution :

Here is an algorithm for the program:

1. Get the employee's annual gross pay.
2. Get the amount of bonuses paid to the employee.
3. Calculate and display the contribution for the gross pay.
4. Calculate and display the contribution for the bonuses.

Chapter 3 – Modules




```
 1 // Global constant for the rate of contribution.
 2 Constant Real CONTRIBUTION_RATE = 0.05
 3
 4 // main module
 5 Module main()
 6     // Local variables
 7     Declare Real annualGrossPay
 8     Declare Real totalBonuses
 9
10     // Get the annual gross pay.
11     Call getGrossPay(annualGrossPay)
12
13     // Get the total of the bonuses.
14     Call getBonuses(totalBonuses)
15
16     // Display the contribution for
17     // the gross pay.
18     Call showGrossPayContrib(annualGrossPay)
19
20     // Display the contribution for
21     // the bonuses.
22     Call showBonusContrib(totalBonuses)
23 End Module
24
25 // The getGrossPay module gets the
26 // gross pay and stores it in the
27 // grossPay reference variable.
28 Module getGrossPay(Real Ref grossPay)
29     Display "Enter the total gross pay."
```

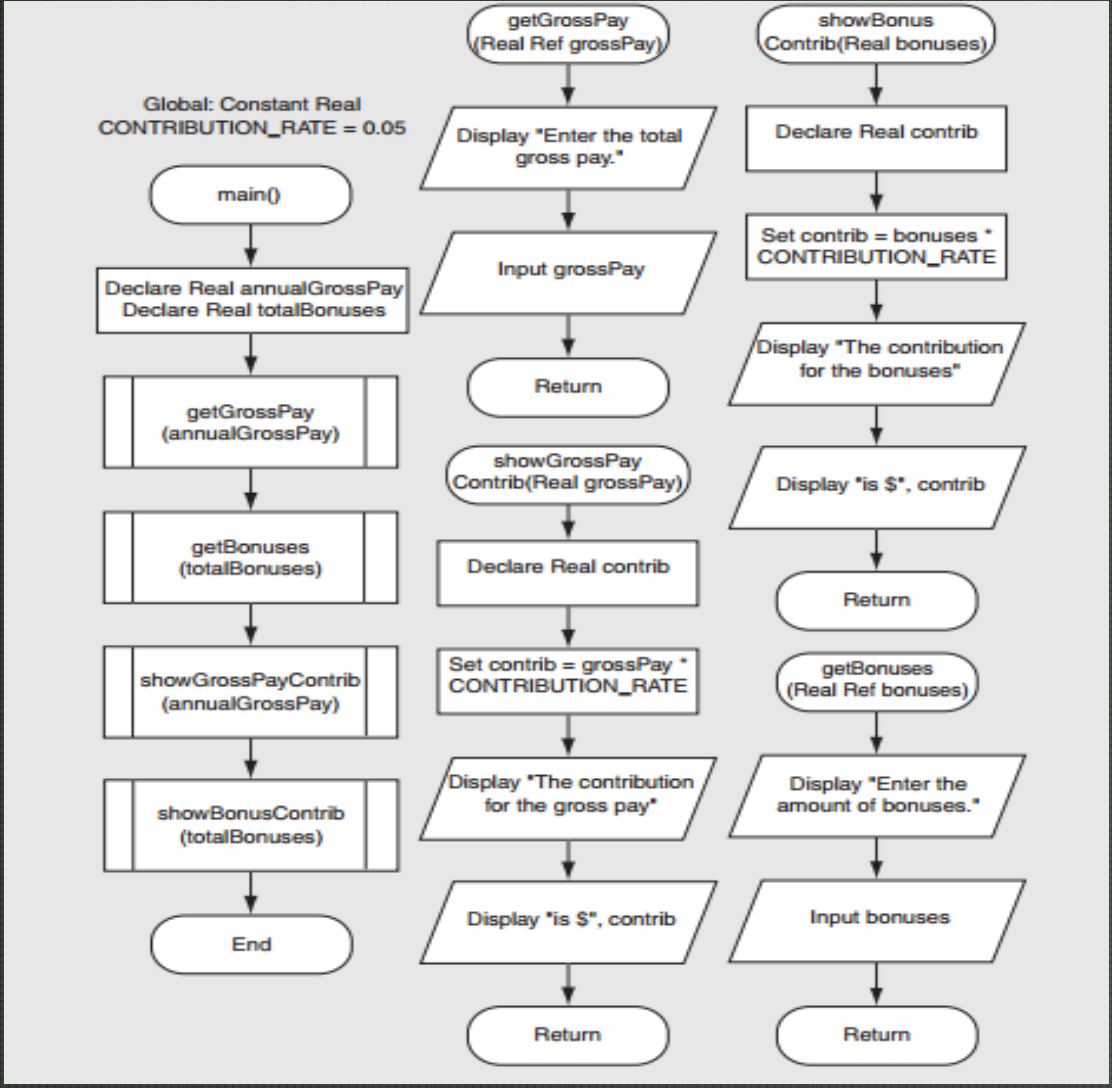
Chapter 3 – Modules

```
30    Input grossPay
31 End Module
32
33 // The getBonuses module gets the
34 // amount of bonuses and stores it
35 // in the bonuses reference variable.
36 Module getBonuses(Real Ref bonuses)
37    Display "Enter the amount of bonuses."
38    Input bonuses
39 End Module
40
41 // The showGrossPayContrib module
42 // accepts the gross pay as an argument
43 // and displays the retirement contribution
44 // for gross pay.
45 Module showGrossPayContrib(Real grossPay)
46    Declare Real contrib
47    Set contrib = grossPay * CONTRIBUTION_RATE
48    Display "The contribution for the gross pay"
49    Display "is $", contrib
50 End Module
51
52 // The showBonusContrib module accepts
53 // the bonus amount as an argument and
54 // displays the retirement contribution
55 // for bonuses.
56 Module showBonusContrib(Real bonuses)
57    Declare Real contrib
58    Set contrib = bonuses * CONTRIBUTION_RATE
59    Display "The contribution for the bonuses"
60    Display "is $", contrib
61 End Module
```

Program Output (with Input Shown in Bold)

```
Enter the total gross pay.
80000.00 [Enter]
Enter the amount of bonuses.
20000.00 [Enter]
The contribution for the gross pay
is $4000
The contribution for the bonuses
is $1000
```

Chapter 3 – Modules



Chapter 3 – Modules

Check Point :

What is the scope of a global variable?

Give one good reason that you should not use global variables in a program.

What is a global constant? Is it permissible to use global constants in a program?

What is a local variable? How is access to a local variable restricted?

What is a variable's scope?

Is it usually permissible to have more than one variable with the same name in same scope? Why or why not?

Is it usually permissible for a local variable in one module to have the same name as a local variable in a different module?

Chapter 3 – Modules

Programming Exercises : (Pseudocode /Flowchart)

1.Stadium Seating

There are three seating categories at a stadium. For a softball game, Class A seats cost $15, Class B seats cost $12, and Class C seats cost $9. Design a modular program that asks how many tickets for each class of seats were sold, and then displays the amount of income generated from ticket sales.

2.How Much Insurance?

Many financial experts advise that property owners should insure their homes or buildings for at least 80 percent of the amount it would cost to replace the structure. Design a modular program that asks the user to enter the replacement cost of a building and then displays the minimum amount of insurance he or she should buy for the property.

3.Property Tax

A county collects property taxes on the assessment value of property, which is 60 percent of the property's actual value. For example, if an acre of land is valued at $10,000, its assessment value is $6,000. The property tax is then 64¢ for each $100 of the assessment value. The tax for the acre assessed at $6,000 will be $38.40. Design a modular program that asks for the actual value of a piece of property and displays the assessment value and property tax.