CYPRUS INTERNATIONAL UNIVERSITY
ENGINEERING FACULTY

Lecture 2A

Programming Structure

CMPE223 / ISYE223
ALGORITHMS AND PROGRAMMING
2023 – 2024 Spring

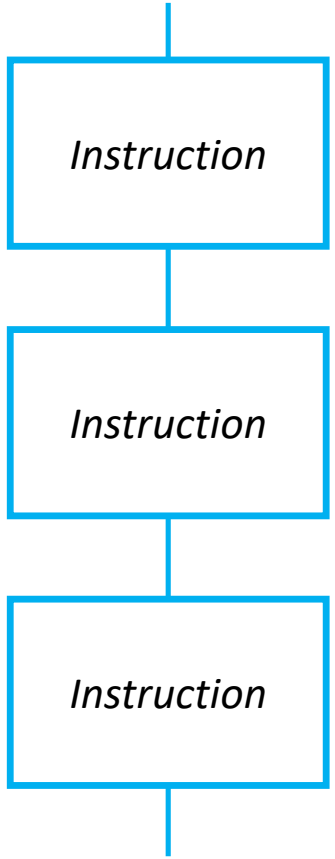
Guidelines for Structuring Programs

To develop efficient computer solutions to problems:

- Use modules to break the whole into parts, each having a particular function (this also eliminates rewriting of identical processes).
- Use the three logic structures to ensure the solution flows smoothly from one instruction to the next.
    1. The sequential structure
    2. The decision structure
    3. The Loop Structure
- Use techniques to improve readability, such as proper naming of variables, internal documentation, and proper indentation.

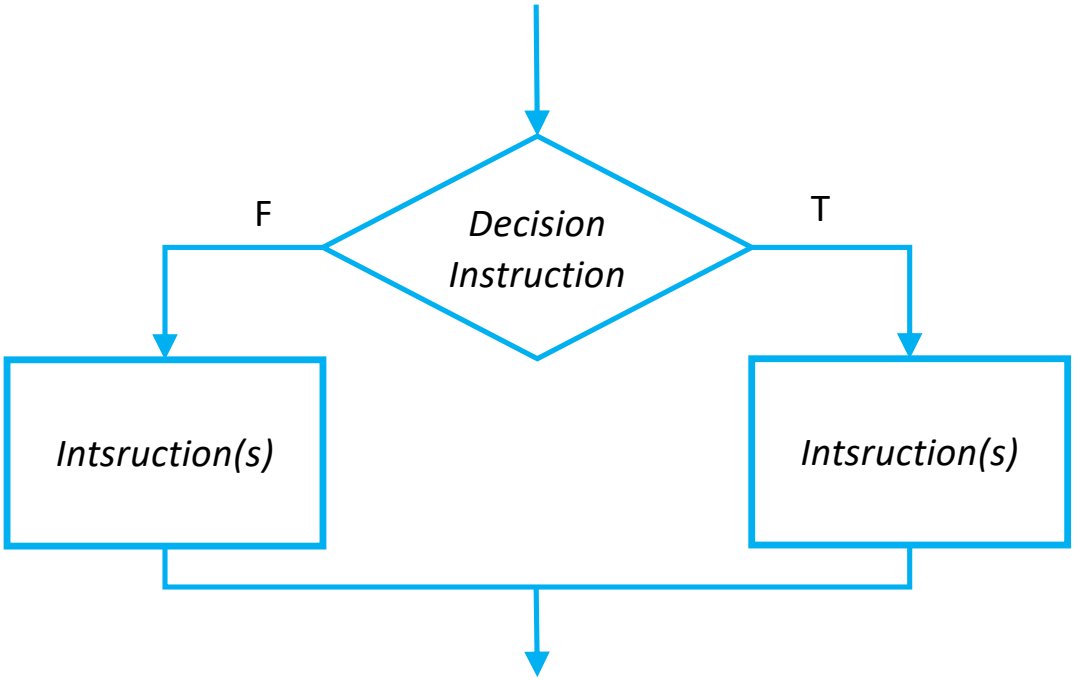
1. Sequential Structure

- Executes instructions one after the other sequentially from the top to bottom.



2. Decision Structure

- Branches to execute one of two possible sets of instructions.
- ***Decision Instruction*** tests a condition.
- The instructions on the **T** branch are processed when the result of the condition is True, and instructions on the **F** branch are processed when the result of the condition is False.
- Uses the **If**/**Then**/**Else** instruction (**If** a condition is **True**, **Then** execute a set of instructions, or **Else** execute another set of instructions).



2. Decision Structure (cont'd)

A condition can be one of four things:

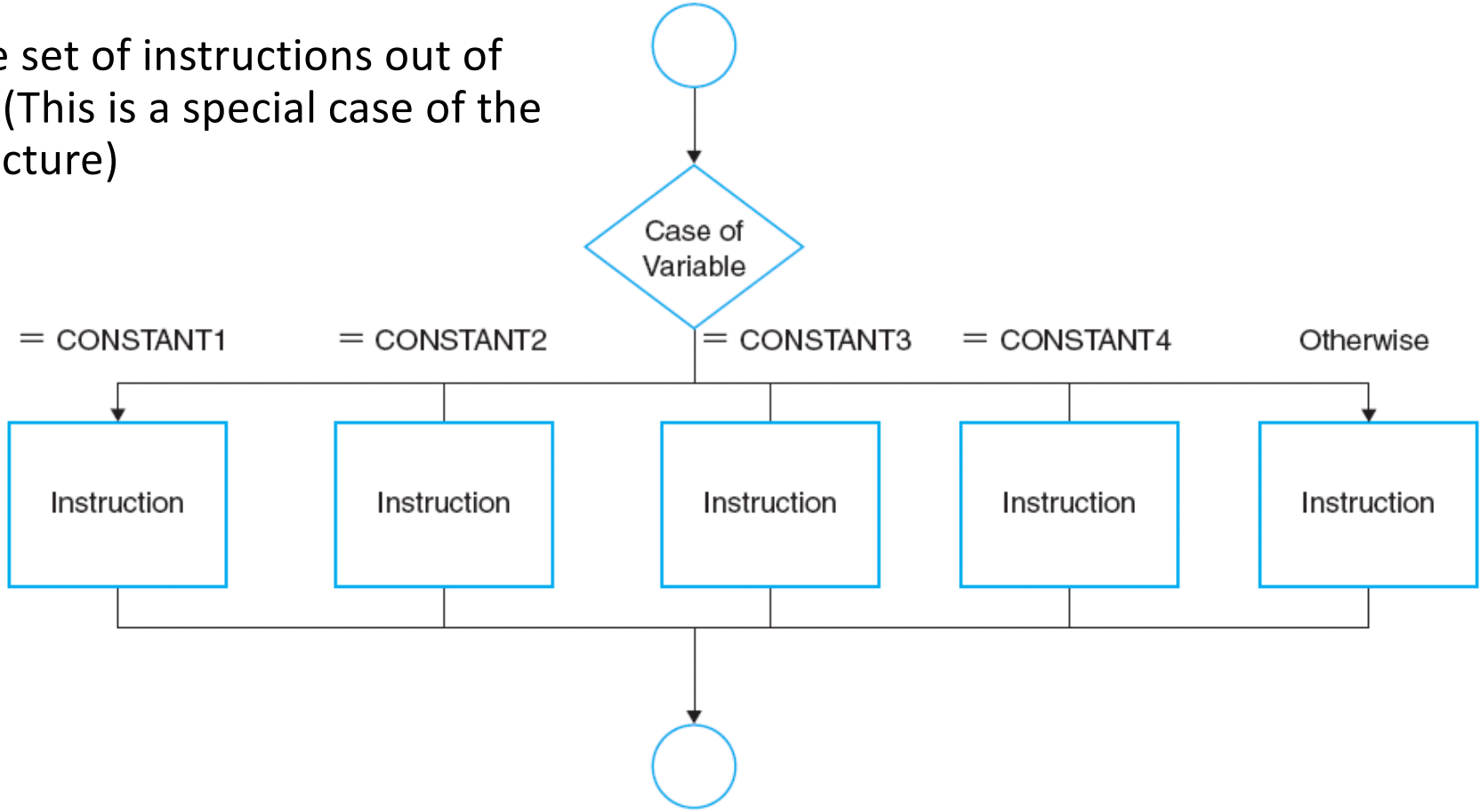
- A logical expression, that is, an expression that uses logical operators (AND, OR, and NOT).
- An expression using relational operators (greater than, less than, greater than or equal to, less than or equal to, equal to, and not equal to).
- A variable of the logical data type (True or False).
- A combination of logical, relational, and mathematical operators.

Case Logic Structure

- The case logic structure is a special variation of the decision structure.
- Sometimes, a solution to a problem requires the computer to select one action from a set of actions. This kind of solution can be designed through the decision logic structure, but the case logic structure is more efficient.
- The case logic structure comprises several sets of instructions, usually only one of which will be selected by a condition and then executed by the computer.
- The case logic structure does not enable the program to loop back to select another option. A loop structure must enclose the case structure if the solution requires looping back.

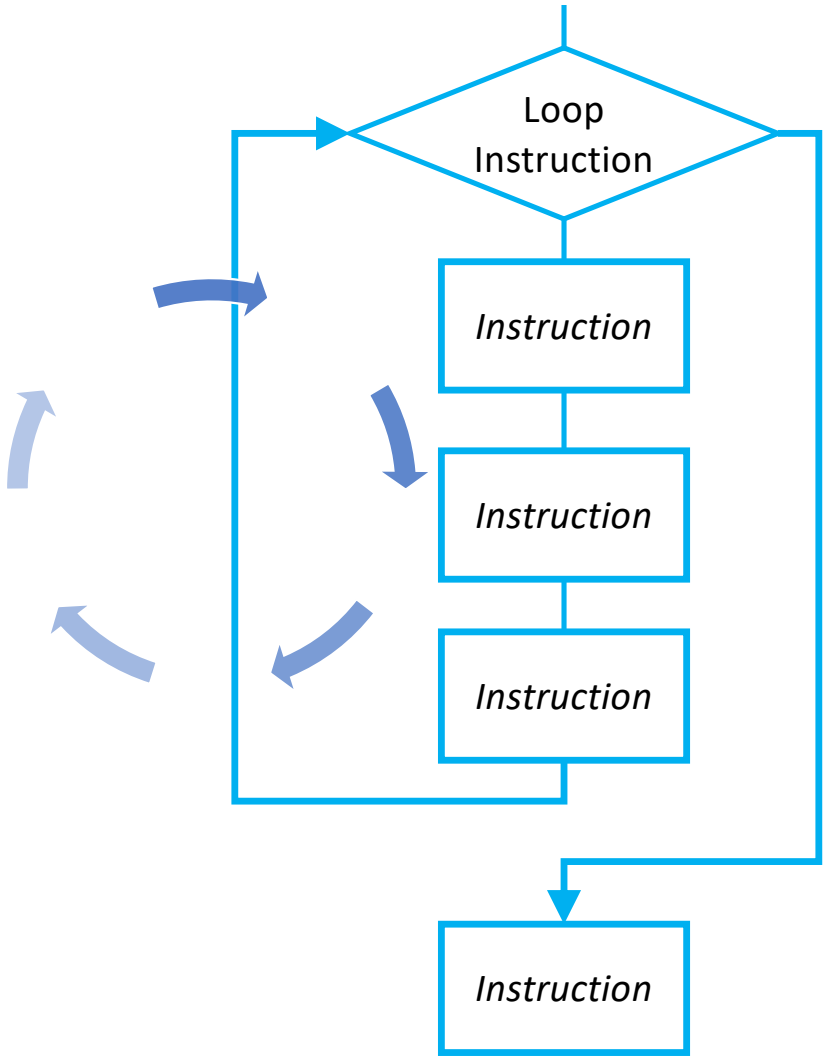
Case Logic Structure (cont'd)

- Executes one set of instructions out of several sets. (This is a special case of the decision structure)



3. Loop Structure

- The loop structure is used for repeating the same task over and over for different sets of data.



3. Loop Structure (cont'd)

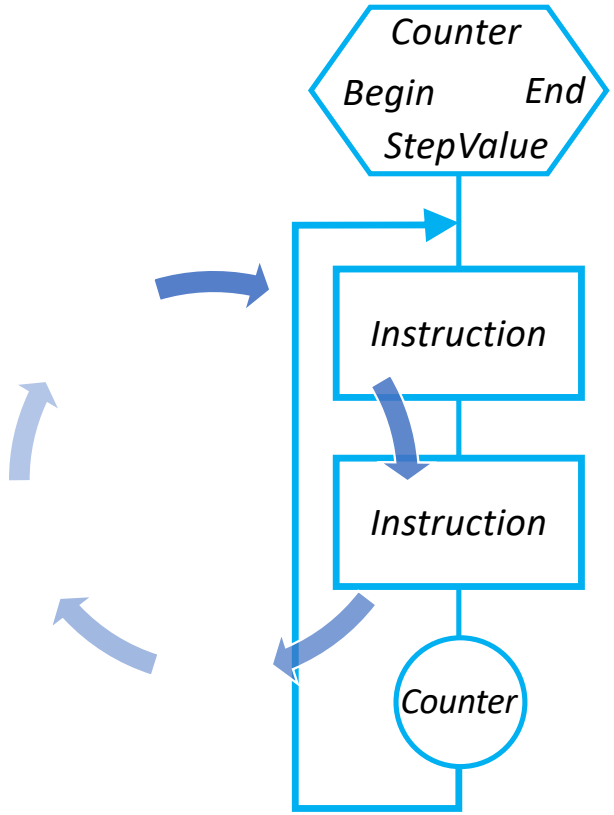
There are three types of loop structures

1. The **While** loop repeats instructions while a condition is True and stops repeating when a condition arises that is not True.
2. The **Repeat/Until** loop repeats instructions while a condition is False or until a condition is True.
3. The **automatic counter** loop: A variable is set equal to a given number and increases in equal given increments until it is greater than an ending number.

- Each of the three loop structures has a specific use according to the language and/or the problem.
- The algorithm and the flowchart differ with each type of loop structure.
- It is essential to indent the instructions in the loop structure to improve readability.

3. Loop Structure (cont'd)

- The **automatic-counter** loop.



Recursion

- Another type of loop structure is *recursion*.
- Recursion occurs when a module (function) calls **itself**.
- The condition that ends the loop must be within the module.
- Recursion usually works faster than conventional loop structures.

Guidelines for Designing Modules

- A module controls the order of processing of a single function, such as printing, calculating, or entering data.
- There is one entrance and one exit to a module (the processing starts at the top and ends at the bottom).
- Each module must be short enough to be easily read and modified.

Types Of Modules

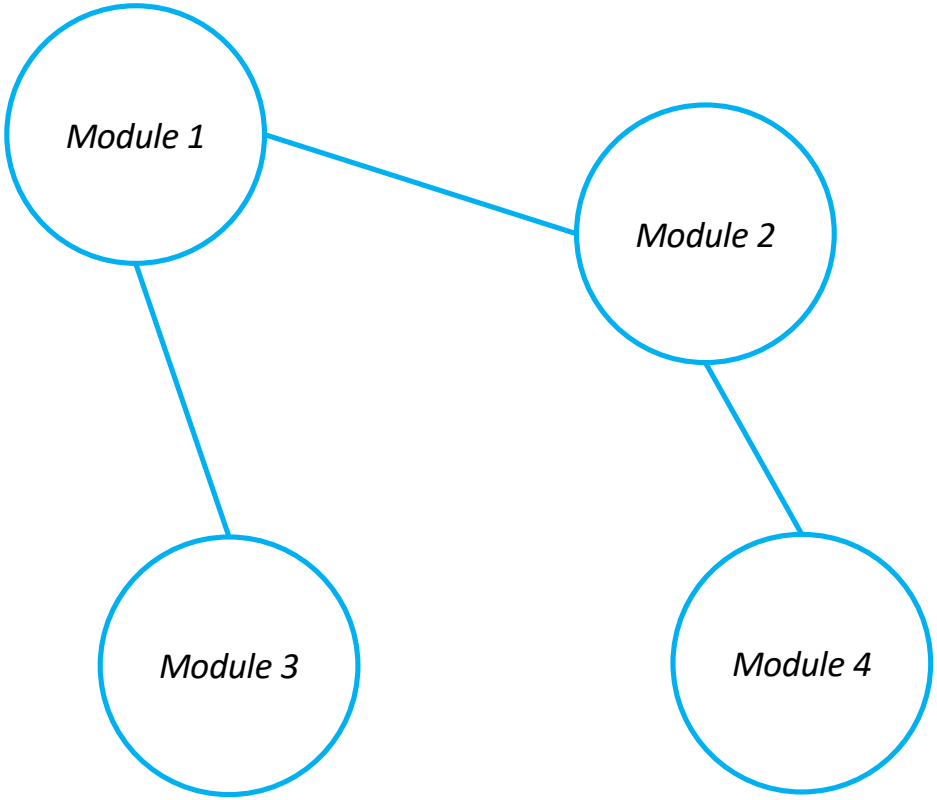
- The *Control* module shows the overall flow of the data through the program. All other modules are subordinate to it.
- The *Initialisation* module processes instructions that are executed only once during the program, and only at the beginning.
  - These instructions include opening files and setting the beginning values of variables used in processing.
- *Wrap-up* modules process all instructions that are executed only once during the program and only at the end. These instructions include closing files and printing totals, among others.

Types Of Modules

- The *Process* modules may be processed only once, or they may be part of a loop, which is processed more than once during the solution. There are several kinds of Process modules:
  - *Calculation* modules do arithmetic calculations, accumulating (summing to calculate totals), counting, or manipulating numerical data in some other way, such as putting numbers in numerical order. Sometimes Calculation modules manipulate string data to perform tasks such as putting a list of values in alphabetical order.
  - *Print* modules print output lines (the results of the processing, line by line), including headings and summaries.
  - *Read* and *Data Validation* modules read or input the data into the internal memory of the computer, validate data (check the accuracy of the data), enter data from the keyboard, and so forth. Validation modules are usually separate from Read modules.

Cohesion and Coupling

- Modules should be functionally independent and perform a single task (*cohesion*).
- Cohesion is the ability of each module to be independent of other modules.
- However, modules must be connected (*coupling*) to allow data sharing.
- High cohesion and low coupling usually indicate a well-structured design.



Cohesion

- *Cohesion* is the ability of a module to **work independently** from all other modules.
- Each module should have a single entry and exit to work independently.
- Cohesion allows us to write a module and test it independently. This module can then be used in a more than one program (which effectively makes it a “black box”, we need not worry about what the instructions are as long as a correct value is returned).
- Cohesion is a measurement of how closely related the functions are.
- High cohesion is desirable because modules written with high cohesion tend to be more understandable, more reliable, and reusable.

Coupling

- Modules may need to share data from other modules. This sharing of data is called *coupling*.
- Coupling is accomplished by some type of interface between modules that enables data to be passed from one module to another with the minimum interruption of modular independence.
- Coupling allows communication between modules.

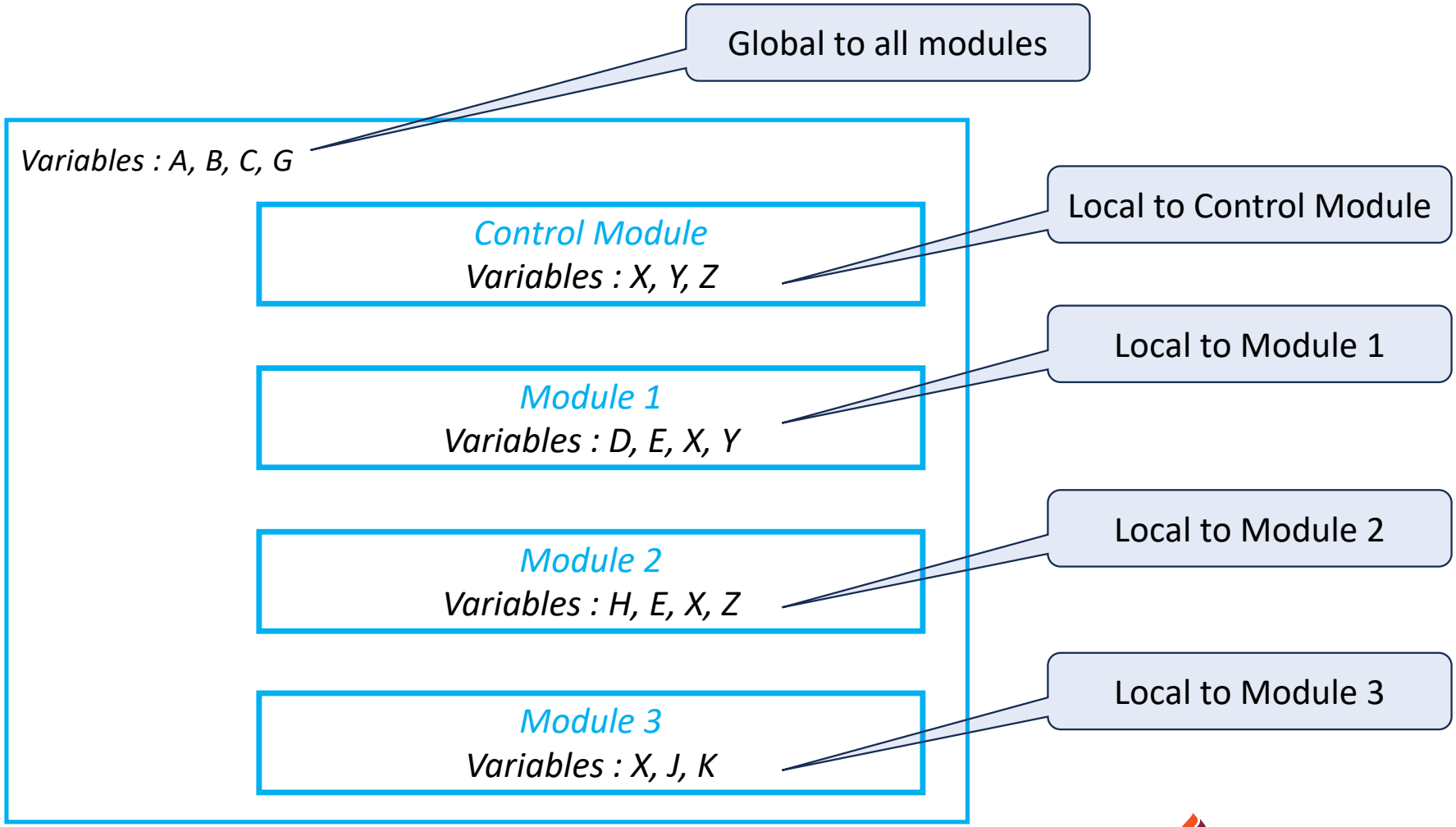
Ways to Accomplish Coupling

- The three ways to couple modules are through the use of:
  1. Global variables
  2. Parameters
  3. Return value

1. Local and Global Variables

- *Local variables* are defined within each module and can only be used by the module itself. Other modules do not know the existence of those variables; hence, they can not access them. This allows cohesion.
- *Global variables* are defined outside of all the modules. hence all modules can see and access global variables. They are global to the program and this allows coupling.
- The significant difference between the two is the *scope* of the variables.
  - Scope designates where the variable's value can be used—the visibility of a variable by different modules.

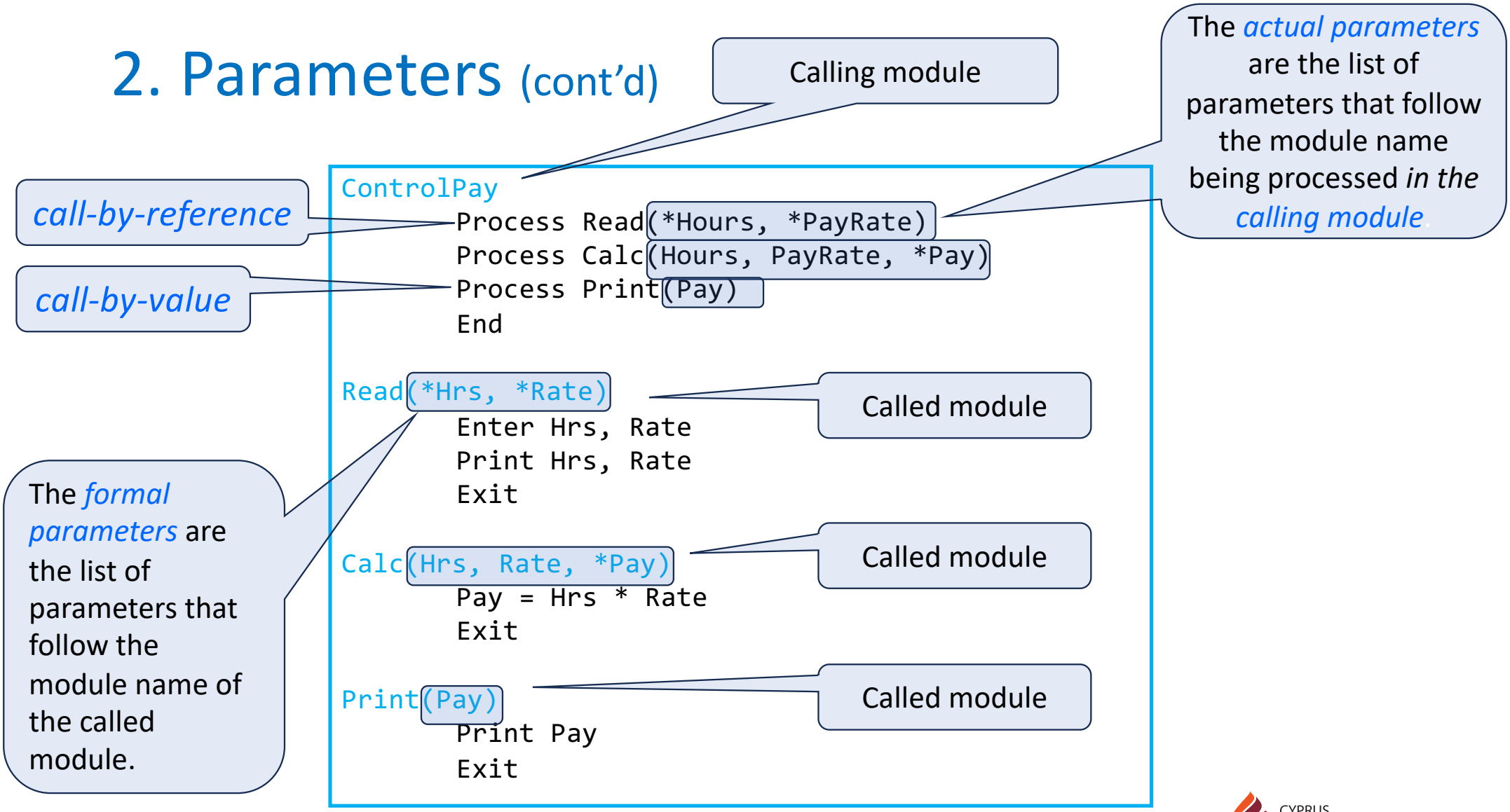
Scope of Variables



2. Parameters

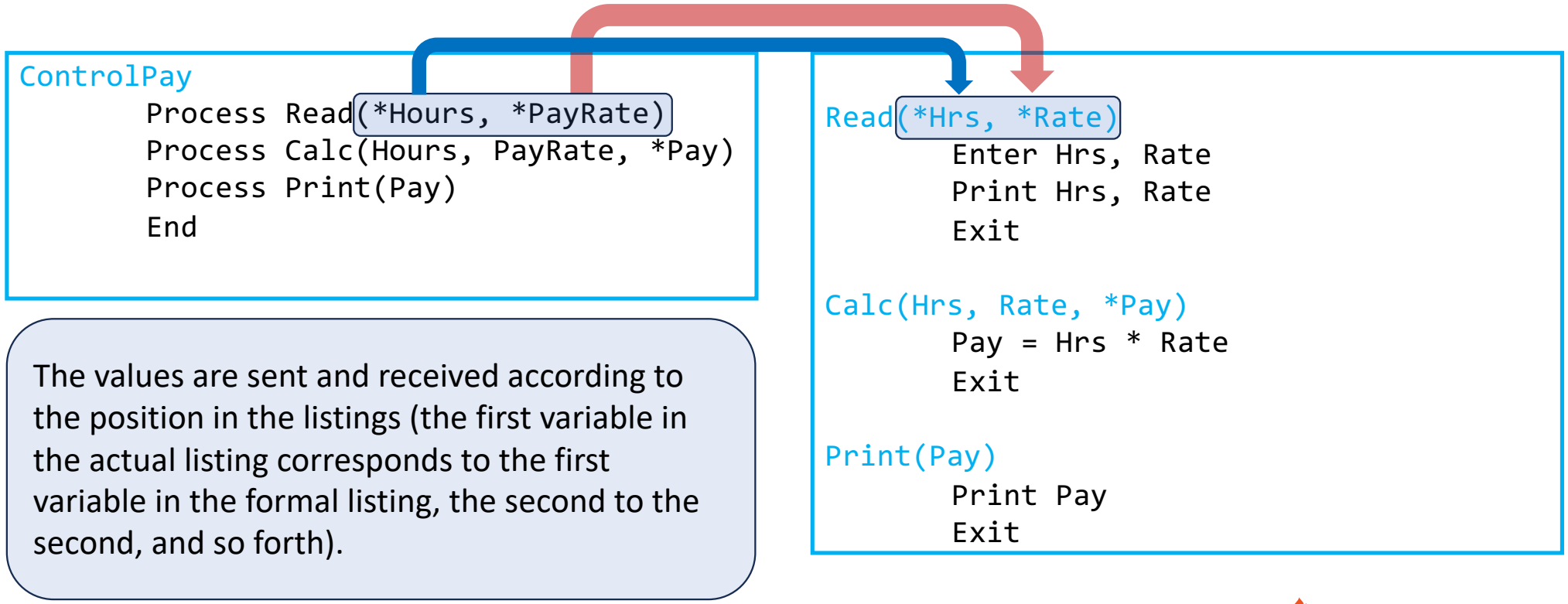
- *Parameters* are local variables that are sent from one module to another (a way of facilitating coupling that allows the communication of data between modules).
- They are placed after the module (function) name and within parenthesis, for example: Read(A, B, C).
- There are two ways to send data from one module to another through the use of parameters.
  - The first is to send a copy of the variable's value, *call-by-value* parameter.
  - The second is to send the *address* of a variable, *call-by-reference* parameter specified by using an asterisk (*) in front of the variable name.

2. Parameters (cont'd)



2. Parameters (cont'd)

- Notice that the variable names in the actual parameter listing may or may not be the same as those in the formal parameter listing.



The values are sent and received according to the position in the listings (the first variable in the actual listing corresponds to the first variable in the formal listing, the second to the second, and so forth).

Parameters (cont'd)

- Using parameters is one of the best methods of coupling modules.
- Parameters in the formal parameter listing are neither local nor global variables and therefore should not be declared as local or global variables.
- In the event of variable name duplication, the local variable has precedence over the parameter variable, and the parameter variable has precedence over the global variable.
  - When the program looks for the variable name, it first looks at the local variable list, then at the formal parameter list, and last at the global variable list. An error occurs if it is not found in any of the three lists.

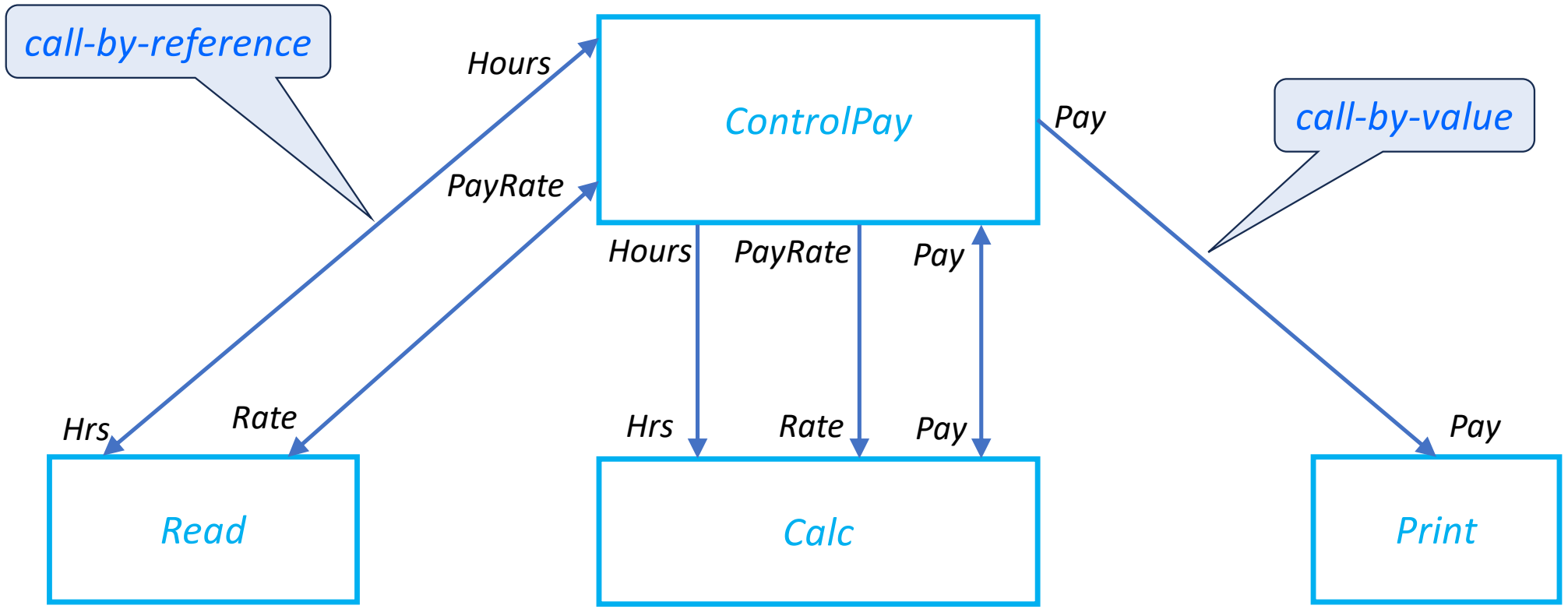
3. The Return Value

- The return value is the result of a function.
- At the conclusion of the function's execution, the result is placed temporarily in the function name.
- The return value is only sent from the called module into the calling module.
- The value of the function's name will no longer be available once the instruction has been executed.
- The next call to the same function will return a new result within the function name.

Coupling Diagram

- A coupling diagram indicates how data couples modules together.
- A coupling diagram has a rectangle for each module and a line connecting the modules for each parameter sent from one module to another.
- The double-headed arrows indicate that these parameters are call-by-reference parameters and changes can be seen by both modules.
- Single-headed arrows indicate that these are call-by-value parameters and changes can be seen only by the called module.

Coupling Diagram (cont'd)



References

Sprankle, M., & Hubbard, J. (2008). *Problem solving and programming concepts*. Prentice Hall Press.