Problem Solving

LOCAL, GLOBAL & PARAMETERS

by
Noor Azida Binti Sahabudin
Faculty of Computer Systems & Software Engineering
azida@ump.edu.my

Chapter Description

- Aims
  - A
- Expected Outcomes
  - Explain the difference between **local** and **global** variables
  - Explain the use of **parameters**
- References
  - Sprankle, M., and Hubbard, J., (2012). Problem Solving and Programming Concepts : 9th Edition. Prentice Hall, 2012. ISBN : 0132492644

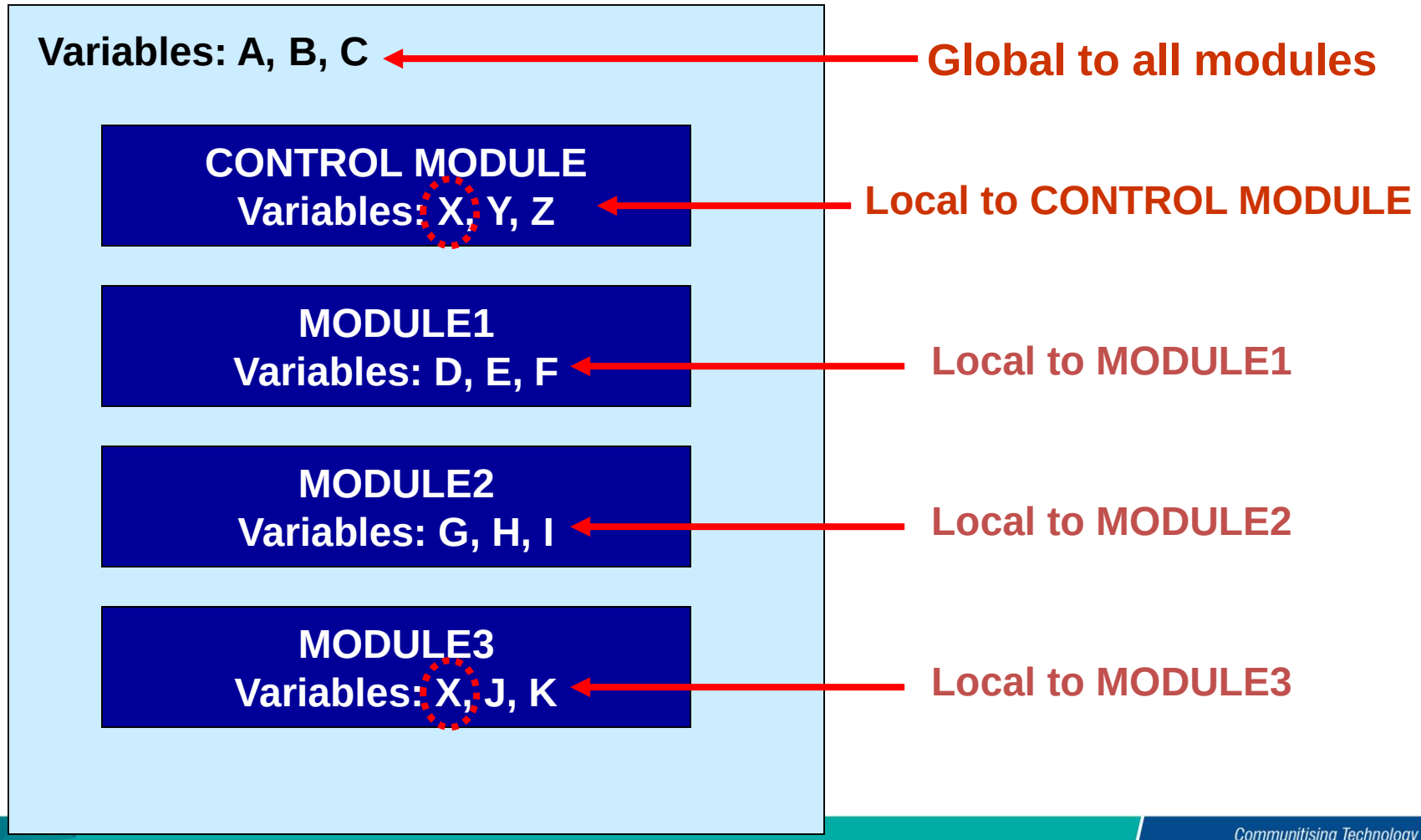
Local and Global Variable

- **Local Variable**
  - Variable that defined **within** a module
  - **Unknown** to other module and to the main program
  - Used **only** by the module itself
  - Allows **cohesion** to take place
  - Programmer does not have to worry about variable name duplication in modules created by other programmers
  - If other modules need to use the value of a variable, the modules must be coupled through the use of parameters or return value

Local and Global Variable

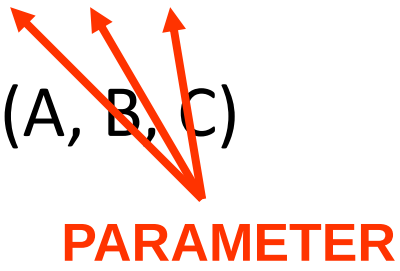
- **Global variable**
  - Variable that are defined **outside** of the **individual's modules**
  - **Global to the program** → visible to all modules
  - Allow data coupling
  - Use for:
    - Variable that will be used in all modules

Local and Global Variable



Parameters

- **Local variables** that are **passed** or **sent** from one module to another
- Another way of **facilitating coupling** that allows the communication of data between modules
- E.g.:
  - READ (A, B, C)



Parameters

- Terminology
  - The **formal** vs. **actual parameter**
  - The **calling module** vs. **the called module**
  - **Call-by-value** vs. **call-by-reference**

Parameters

```
CONTROL PAY
        PROCESS READ (*HOURS, * PAYRATE)
        PROCESS CALC (HOURS, PAYRATE, *PAY)
        PROCESS PRINT (PAY)
        END
READ(*HRS, *RATE)
        ENTER HRS, RATE
        PRINT HRS, RATE
        EXIT
CALC(HRS, RATE, *PAY)
        PAY = HRS * RATE
        EXIT
PRINT (PAY)
        PRINT PAY
        EXIT
```

Calling module

Actual parameter listings

Formal parameter listings
Called modules

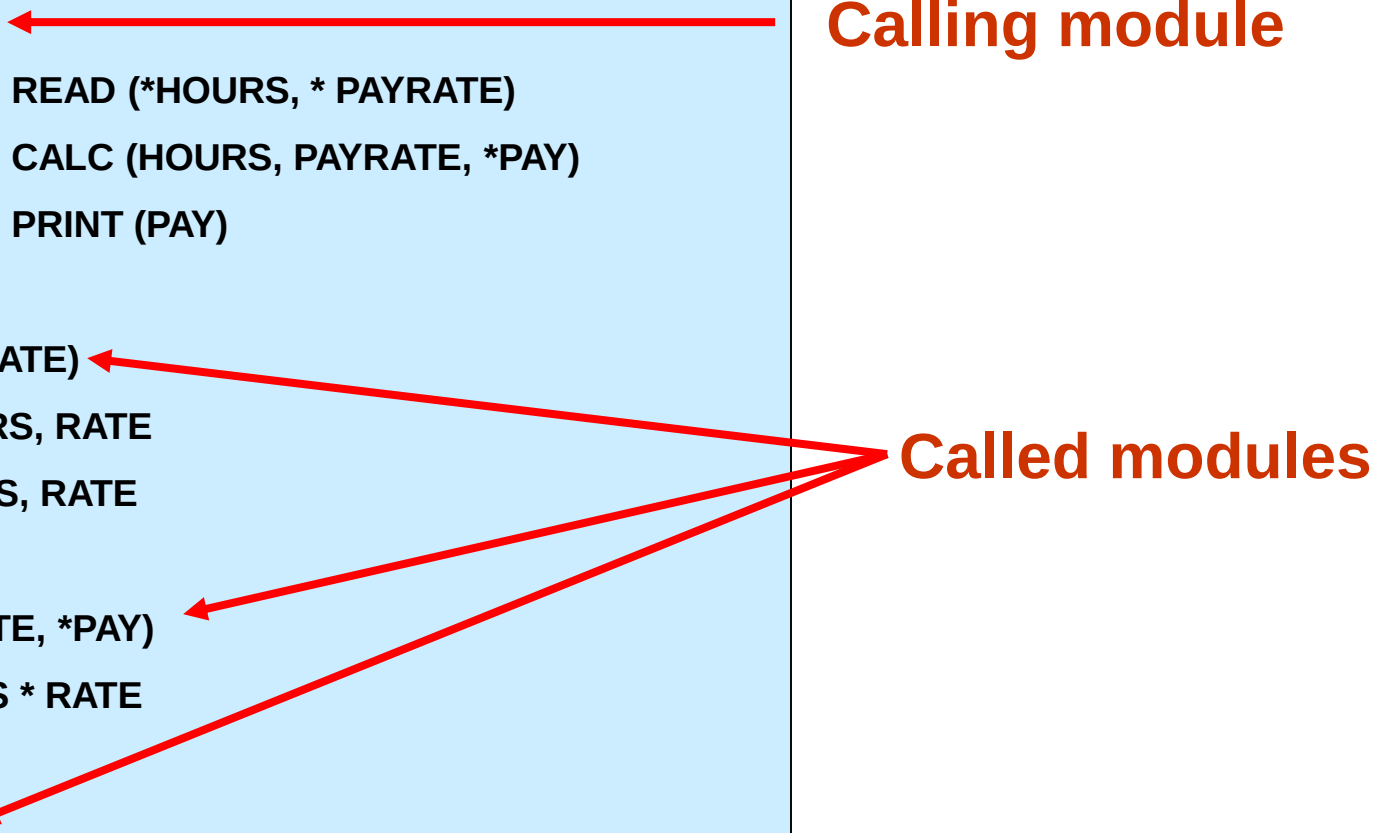
* Indicates call-by-reference parameter, those parameters without an asterisk are call-by-value parameter

Parameters

- **Calling module**
  - Module that **processes another module**
- **Called module**
  - Module that **being** processed

Parameters

```
CONTROL PAY
        PROCESS READ (*HOURS, * PAYRATE)
        PROCESS CALC (HOURS, PAYRATE, *PAY)
        PROCESS PRINT (PAY)
        END
READ(*HRS, *RATE)
        ENTER HRS, RATE
        PRINT HRS, RATE
        EXIT
CALC(HRS, RATE, *PAY)
        PAY = HRS * RATE
        EXIT
PRINT (PAY)
        PRINT PAY
        EXIT
```

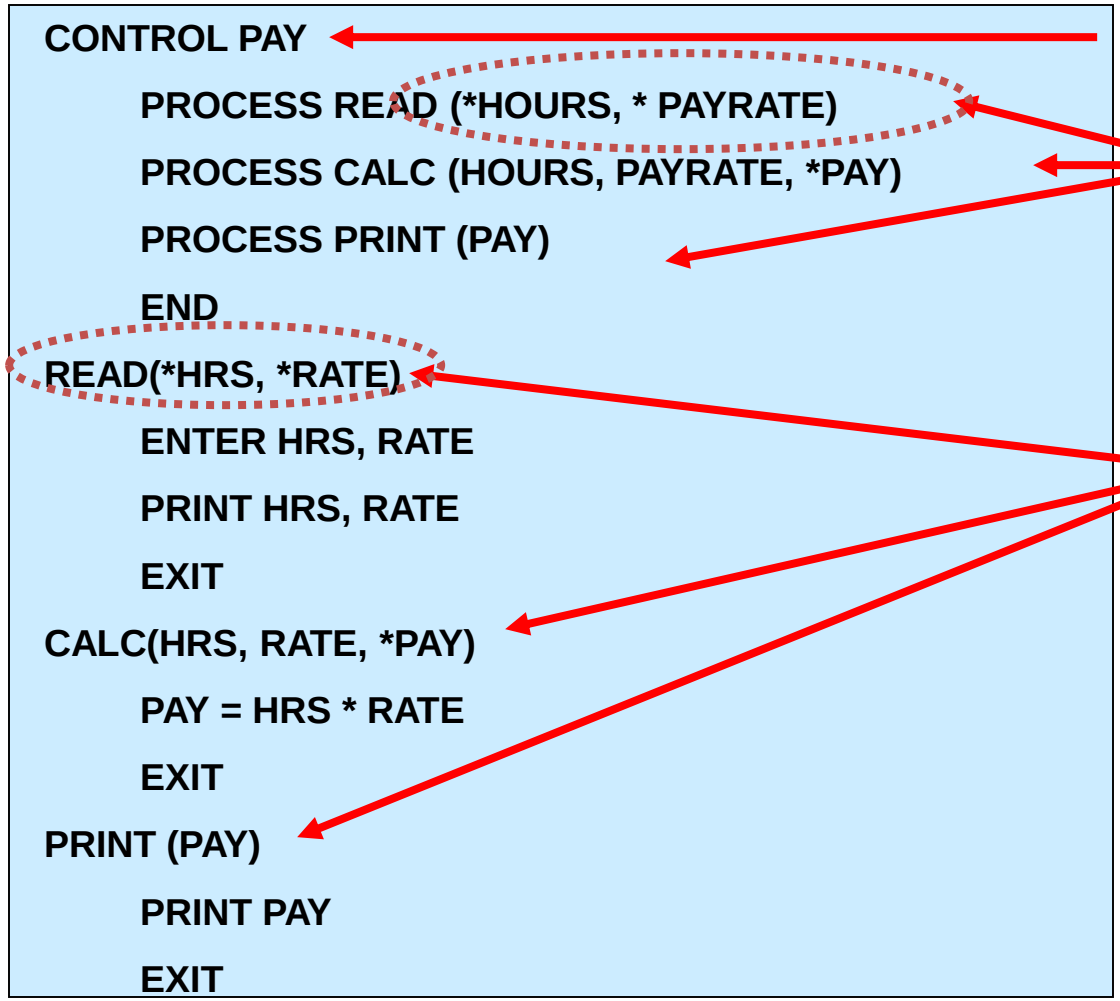


Parameters

- **Actual parameter listing**
  - The list of parameters that follow the **module name being processed** in **calling module**
- **Formal parameter listing**
  - The list of parameters that follow the module name at the **beginning of the module**

Parameters

```
CONTROL PAY
        PROCESS READ (*HOURS, * PAYRATE)
        PROCESS CALC (HOURS, PAYRATE, *PAY)
        PROCESS PRINT (PAY)
        END
READ(*HRS, *RATE)
        ENTER HRS, RATE
        PRINT HRS, RATE
        EXIT
CALC(HRS, RATE, *PAY)
        PAY = HRS * RATE
        EXIT
PRINT (PAY)
        PRINT PAY
        EXIT
```



Return values

- **Return values**
  - The value is **sent out** of the called module into the calling module
  - The result of the function
  - E.g.:
    - *return (variable_name)*

Sending data between modules

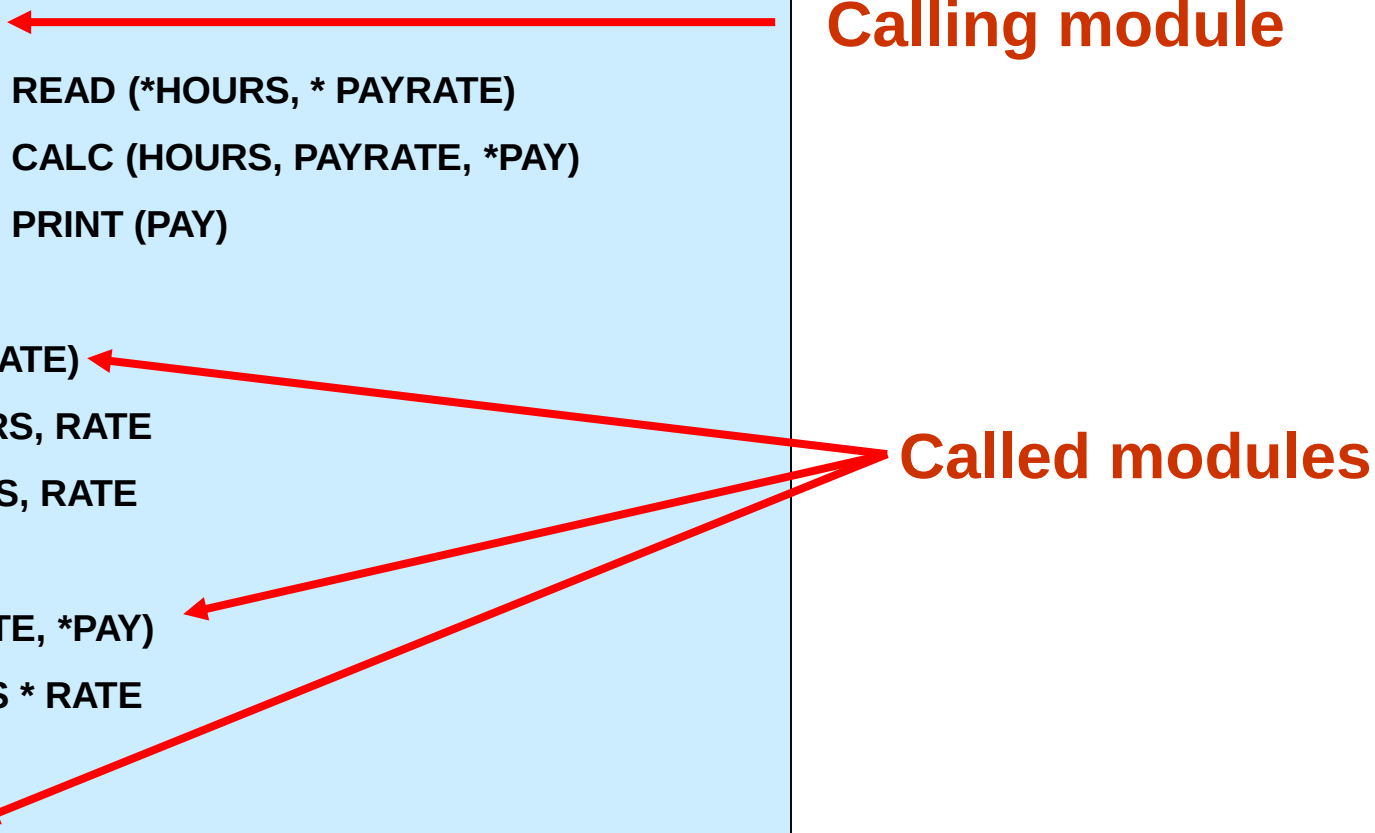
- 2 ways to send data from one module to another module
  - Send **value**
  - Send **address** of variable

Sending data between modules

- **Call-by-value parameter**
  - Send the **value** of the variable
  - the value of the variable is sent to the *called module* by the *calling module*.
  - *called module* will then make a **new memory location** for this variable (since it has no knowledge of where the variable is stored by the calling module)
  - when the *called module* makes a change in the variables, the changes **will not affected** the variable in the *calling module* because the parameter has a different memory location

Parameters

```
CONTROL PAY
        PROCESS READ (*HOURS, * PAYRATE)
        PROCESS CALC (HOURS, PAYRATE, *PAY)
        PROCESS PRINT (PAY)
        END
READ(*HRS, *RATE)
        ENTER HRS, RATE
        PRINT HRS, RATE
        EXIT
CALC(HRS, RATE, *PAY)
        PAY = HRS * RATE
        EXIT
PRINT (PAY)
        PRINT PAY
        EXIT
```



Sending data between modules

- **Call-by-reference parameter**
  - Send the **address** of the variable
  - Specified by the use of an asterisk **(*)** in front of the variable name in both actual and the formal parameter listings
  - the **memory location** is sent, not the value
  - When the *called module* changes the value of the parameter, the *calling module* **will see the change** because they are using the same memory location

Call by-Value and Call by-Reference

CONTROL PAY

1. PROCESS READ (*HOURS, * PAYRATE)

1 **1**

Address of 2000 is sent **Address of 2002 is sent**

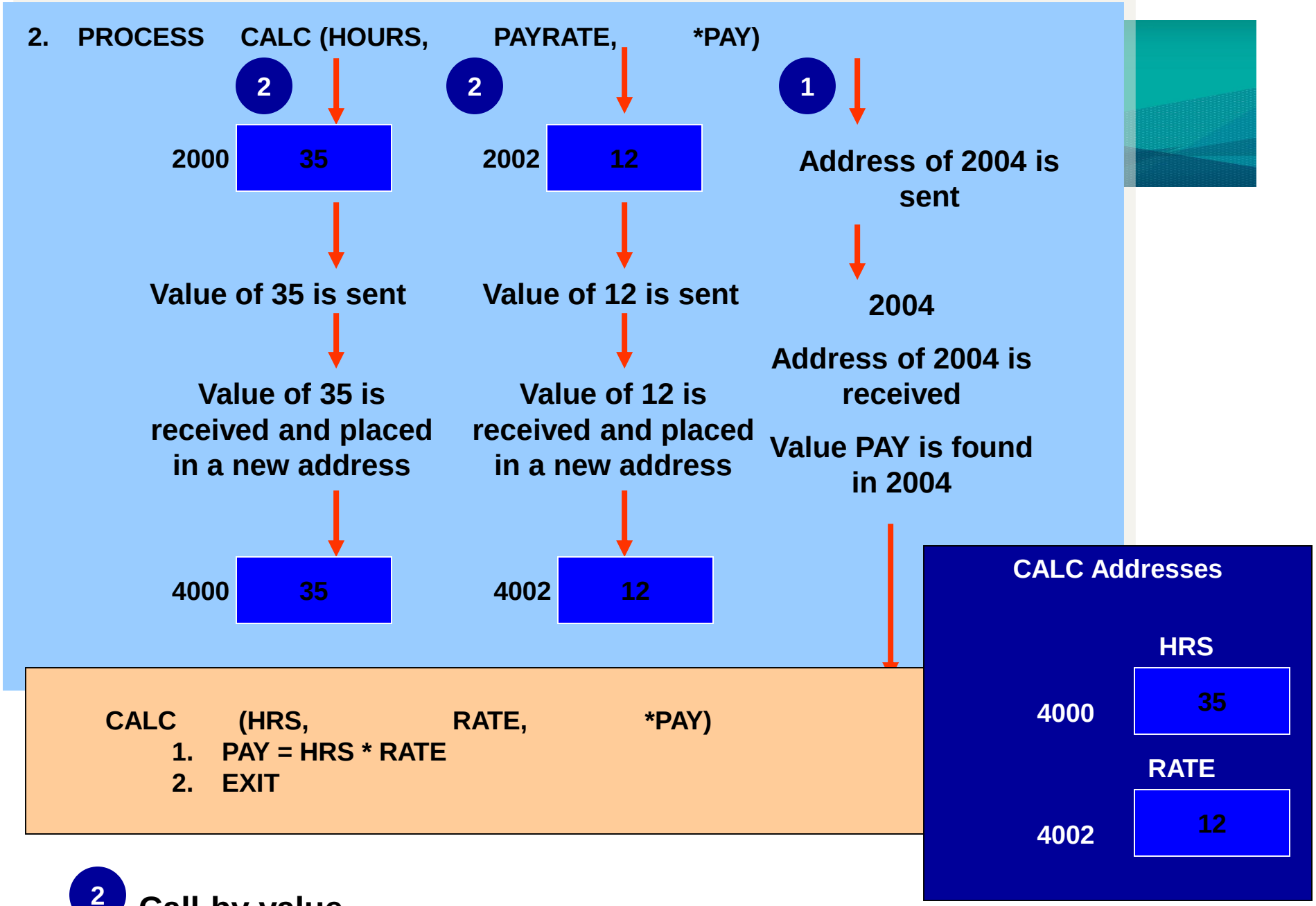
Address of 2000 is received **Address of 2002 is received**

Value of HRS is found in 2000 **Value of RATE is found in 2002**

```
READ (* HRS *RATE)
    1. ENTER HRS, RATE
    2. PRINT HRS, RATE
    3. EXIT
```

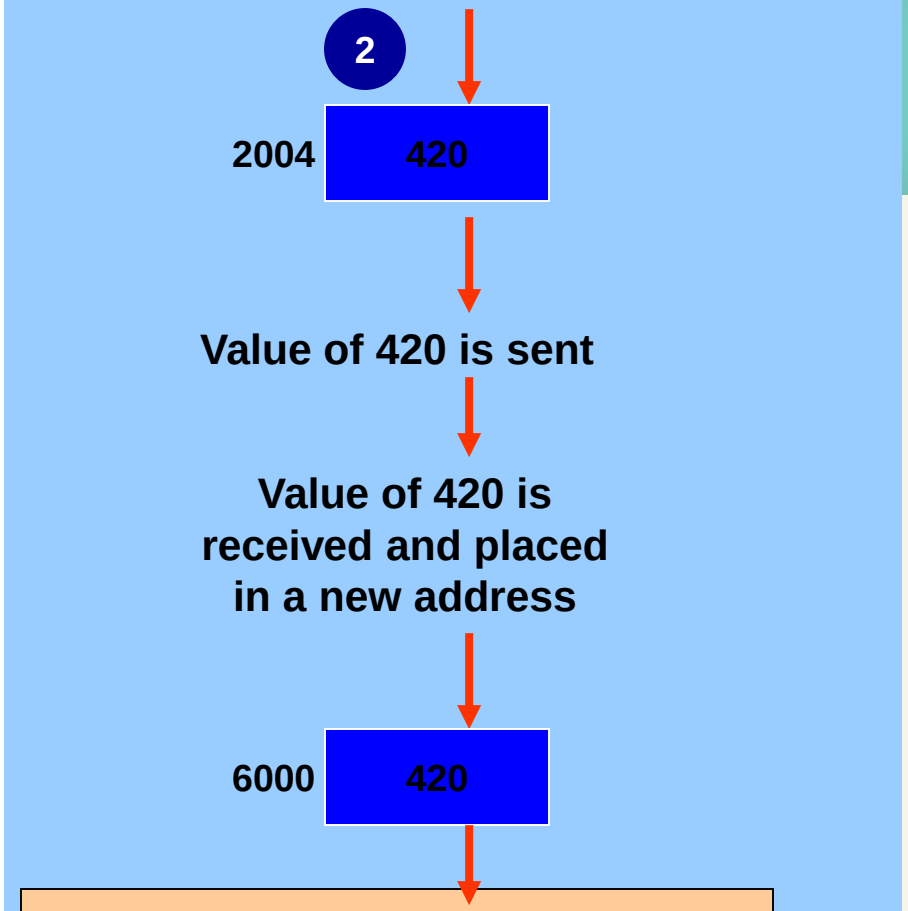
1 **Call-by reference**

2. PROCESS CALC (HOURS, PAYRATE, *PAY)



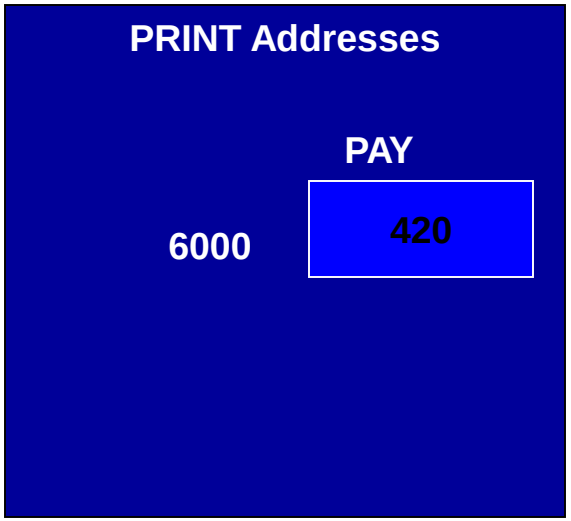
2 Call-by value

3. PROCESS PRINT (PAY)




```
PRINT      (PAY)
      1.   PRINT PAY
      2.   EXIT
```

4. END



Conclusion / What we have learn today?

- Daily life problem
- 6 problem solving steps
- Types of problems (algorithmic vs heuristic)
- Problem solving with computer
- Difficulties with problem solving

Author Information

NOOR AZIDA BINTI SAHABUDIN

Senior Lecturer
Faculty of Computer Systems & Software Engineering
Universiti Malaysia Pahang
PhD in Educational Technology