CYPRUS INTERNATIONAL UNIVERSITY

CYPRUS INTERNATIONAL UNIVERSITY
ENGINEERING FACULTY

Lecture 2

Algorithms, Flowcharts and Interactivity Charts

CMPE223 / ISYE223
ALGORITHMS AND PROGRAMMING
2023 – 2024 Spring

Communicating With The Computer

- The computer must be told what to do.
- Computers do not speak human language; hence, to program a computer, we(programmers) must learn the language of the computer.
- The type of problems that computers can solve is *algorithmic* (problems that can be solved with a sequence of instructions).
- The computer requires instructions written according to syntax rules to understand a message.
- *Syntax* refers to the rules governing the computer's language.

Communicating With The Computer (continued)

- Computers need instructions in the correct sequence to process data to reach the desired results.
- If the instructions are incorrect or not properly sequenced, the computer will give an error message, produce the wrong answer, or give no answer at all.
- Although a set of instructions must be in the correct order to lead to the correct result, there may be several correct orders.
  - Different programmers may develop equally good solutions to a problem, but the solutions may look entirely different.

Programming Concepts

- The solution becomes a *program* when it is written (*coded*) in a computer language.
- An error is called a *bug*.
- The process to find bugs and correcting them is called *debugging*.
- You can find and correct most **logic errors** during the problem-solving process.
- **Syntax errors** can be detected and corrected during coding.

Programming Concepts (continued)

- Three resources need to be optimised for a computer program.
  1. Computer memory
  2. Computer time
  3. Programmer time

Organising The Solution

- To analyse a problem and set up the most efficient solution, we use tools.
- When we do not use these tools during the problem-solving process, the solution takes longer to program, and the final program is less **efficient**, lacks **readability**, and increases programmer frustration.

Tools for Organizing The Solution

1. Analysing The Problem
2. Developing Interactivity Chart
3. Writing The Algorithms
4. Drawing Flowcharts
5. Documenting The Solution

1. Analysing The Problem

- To organise a solution, the first step is to understand and analyse the requirements of the problem.
- Analyse the problem in four parts:
    1. The given data
    2. The required results
    3. The processing that is required in the problem
    4. A list of solution alternatives
- The *Problem Analysis Chart* is a tool that helps to identify the essential data and information in a problem by eliminating the unnecessary words and glean only the facts from the problem.

Problem Analysis Chart (**PAC**)

Given Data	**Required Results**
Data, constants and variables (input values) given in the problem or provided by the user.	Requirements for the output reports. This includes the information needed and the format required.
Processing Required	**Solution Alternatives**
Any equations or other processing required such as sorting, searching, and so forth.	List of alternative ideas for the solution of the problem.

Example

- Calculate the gross pay of an employee given the hours worked and the pay rate.
- The gross pay is calculated by multiplying the hours worked by the pay rate.
- The formula to be used is: Gross Pay = Hours * Pay Rate

Example (continued)

Given Data	Required Results
Hours Pay Rate	Gross Pay
Processing Required	**Solution Alternatives**
Gross Pay = Hours ∗ Pay Rate	1. Define the hours worked and pay rate as constants. 2. Define the hours worked and pay rate as input values.

The second alternative will be used since the program will not need to be changed to calculate another employee's gross pay.

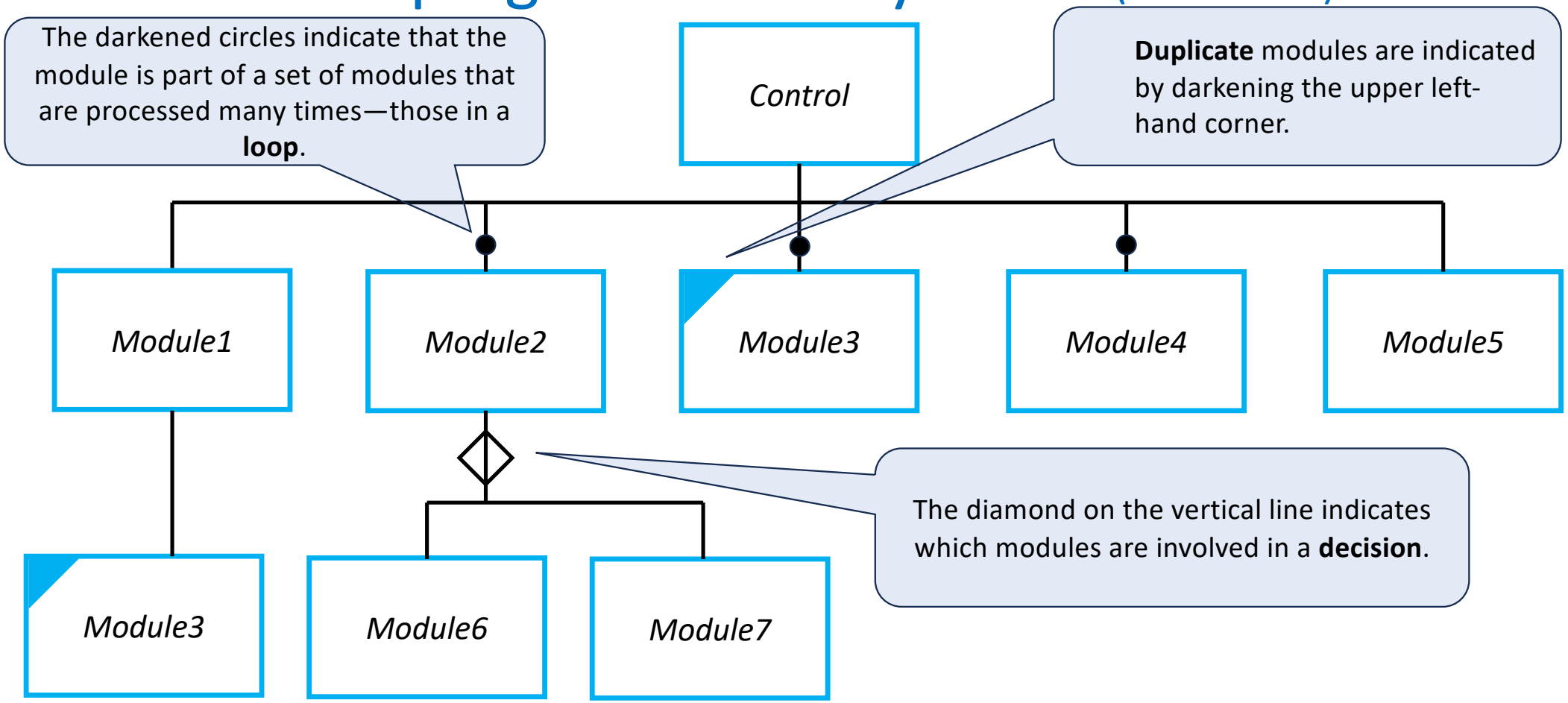
2. Developing Interactivity Chart

- The next step in organising the solution is to divide the process into subtasks called *modules* (functions) and then connect these modules to show their interaction.
- The interactivity chart shows the relationships between these modules.
- Each module should contain the task to accomplish only one *function*, such as entering data, printing result(s), or calculating result(s).
- One *Control Module* (**main module**) will control the program flow to the other modules.
- This breakdown will enable you to view a complex problem in simpler parts and program smaller parts rather than one large, complex program.

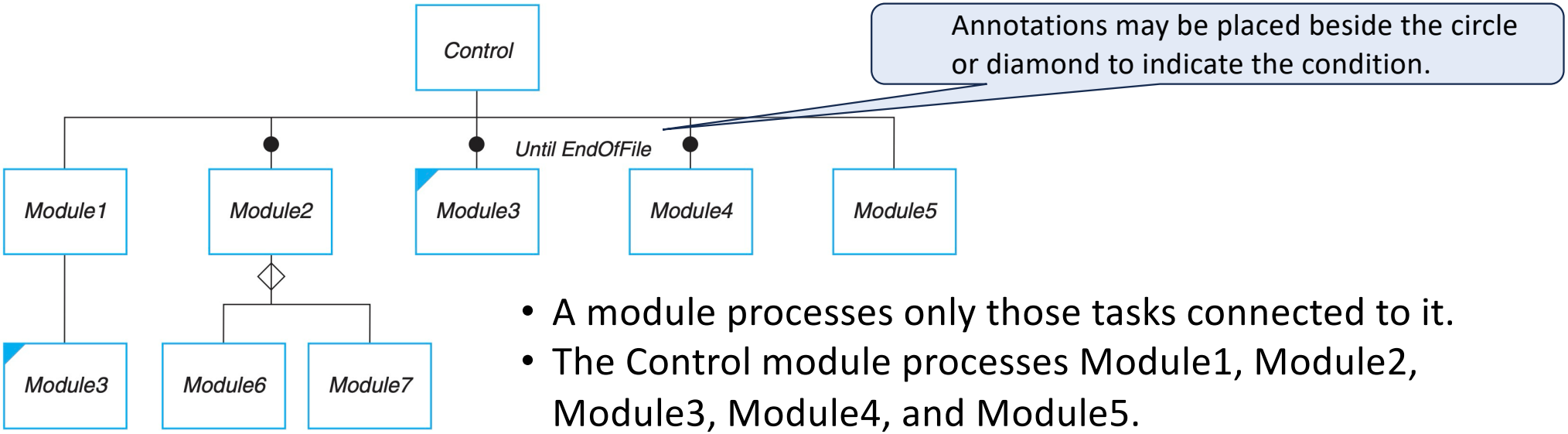
2. Developing Interactivity Chart (continued)

- In developing the Interactivity (Structure) Chart, use the *top-down* method.
  - The top-down method is a procedural programming technique that executes instructions from the first line of code to the end (direct order of processing).
  - The user has little or no control over the order of execution of the modules; the program is completely in control.
- Divide the problem into subtasks and illustrate them in the order they will be processed from the top to the bottom of the chart.
- The module that encompasses the complete solution becomes the *Control Module* since it controls the processing of all the data.
- The subtasks of this module are then located below it.
- The module's name is written inside the rectangle, giving the same name to identical tasks.
- Each module should contain no more than 20 instructions.

2. Developing Interactivity Chart (continued)



2. Developing Interactivity Chart (continued)

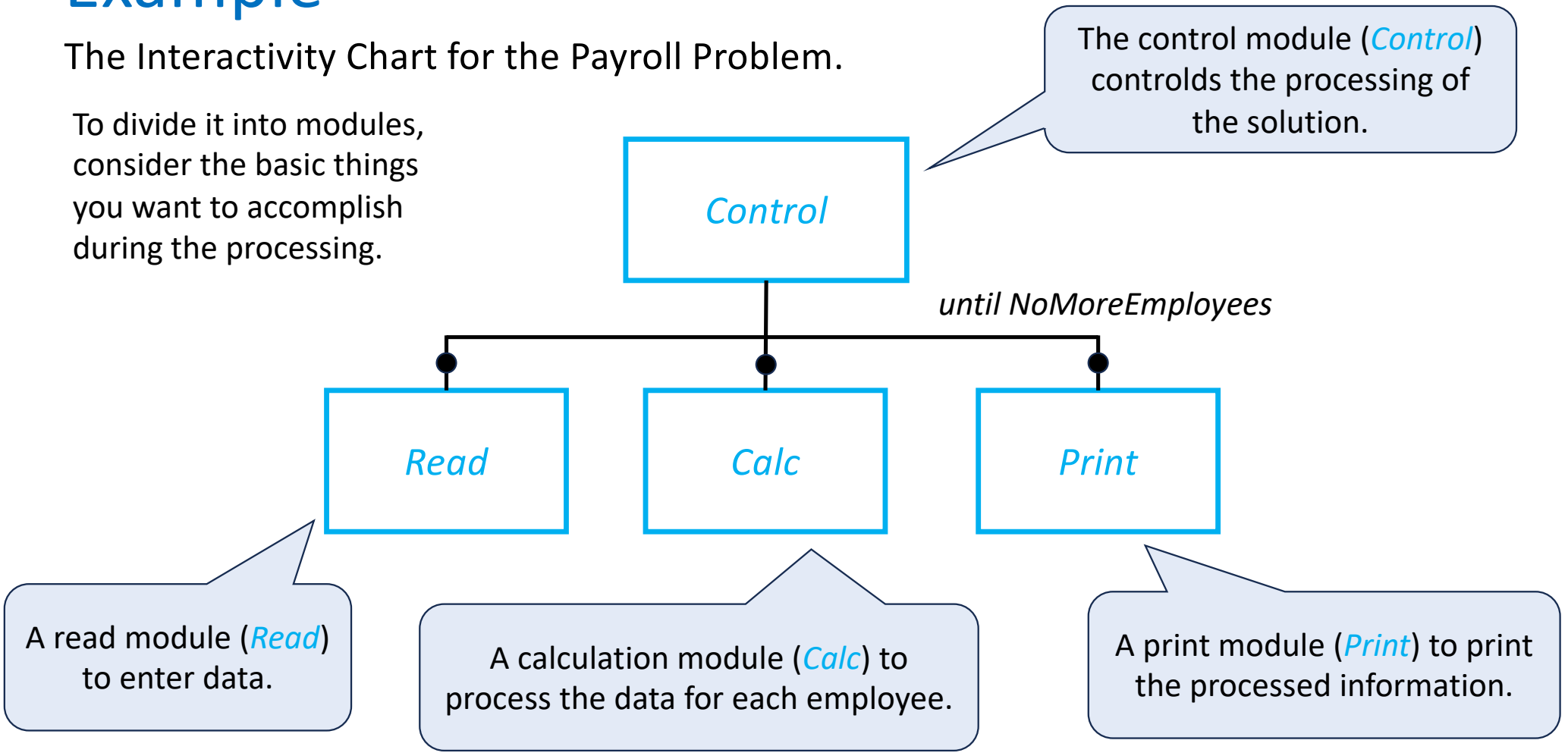


- A module processes only those tasks connected to it.
- The Control module processes Module1, Module2, Module3, Module4, and Module5.
- Module1 processes Module3.
- Module2 processes Module6 and Module7.
  - Only one of these modules will be processed since they are part of a decision.
  - The instructions in Module2 and the appropriate data would lead the computer to choose one module.

Example

The Interactivity Chart for the Payroll Problem.

To divide it into modules, consider the basic things you want to accomplish during the processing.



3. Writing The Algorithms

- The next step in organising a solution is to develop *sets of instructions* called *algorithms*.

An algorithm is a sequence of unambiguous instructions for solving a problem.

3. Writing The Algorithms (continued)

- To complete all of the algorithms needed to solve a problem, a separate set of instructions must be written for each module.
- The instructions:
  - must be written clearly
  - cannot assume anything; must be complete.
  - cannot skip steps
  - each step must be in the correct order
  - must be executable one step at a time

3. Writing The Algorithms (continued)

Control *Module*	*Name of Module (list of parameters)*
1. *Instruction*	1. *Instruction*
2. *Instruction*	2. *Instruction*
3. ..	3. ..
4. ..	4. ..
..	..
—. end	—, exit

Note the Control module uses an end since this is the *end* of the processing.

Other modules use *exit*

3. Writing The Algorithms (continued)

- The algorithms and flowcharts are the final steps in organising a solution.
- Using them, the programmer can test the solution for bugs and go on to code the problem into a computer language.
- A set of data can be tested easily using a flowchart.

3. Writing The Algorithms (continued)

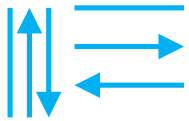
- *Pseudocode* is similar to the algorithm without the numbers and somewhat condensed.
- The pseudocode usually includes some syntax of the language (characteristically closer to what you would write in a computer language).

4. Drawing Flowcharts

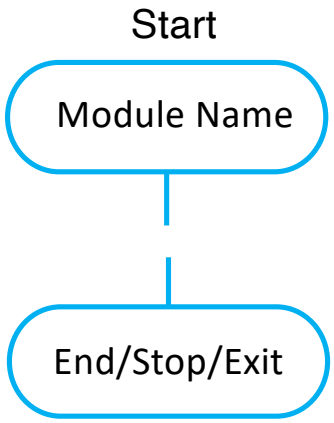
- The flowcharts are developed from the algorithms.
- Flowcharts are graphical representations of the algorithms.
- A flowchart shows the processing flow from the beginning to the end of a solution.
- Each block in a flowchart represents one instruction from the algorithm.
- ISO 5807:1985 (https://www.iso.org/standard/11955.html)
  - Defines symbols to be used in information processing documentation and gives guidance on conventions for their use in data flowcharts, program flowcharts, system flowcharts, program network charts, and system resources charts.

Flowchart Symbols

Flowlines



- Indicate the direction of data flow.
- Flowlines are straight lines with optional arrowheads (only necessary when the flow direction might be in doubt).
- They are used to connect blocks by exiting from one and entering another.



- Flattened ellipses indicate the start and the end of a module.
- An ellipse uses the name of the module at the start.
- The end is indicated by the word "*End*" or "*Stop*" for the Control module and the word "*Exit*" for all other modules.
- Start has no flowlines entering it and only one exiting it.
- End or exit has one flowline entering it but none exiting it.

Flowchart Symbols (continued)

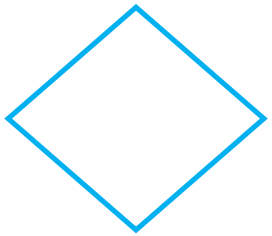
Processing

- The rectangle indicates a processing block, for such things as calculations, opening and closing files, and so forth.
- Processing block has one entrance and one exit.

I/O

- The parallelogram indicates input to and output from the computer memory.
- I/O block has one entrance and one exit.

Decision



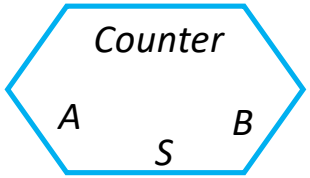
- The diamond indicates a decision.
- It has one entrance and two exits from.
- One exit is the action when the resultant is True and the other exit is the action when the resultant is False.

Flowchart Symbols (continued)

Process Module

- Rectangles with lines down each side indicate the process of modules.
- Have one entrance and one exit.

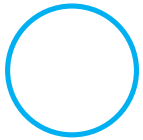
Automatic-Counter Loop



- The polygon indicates a loop with a counter.
- The counter starts with A (the beginning value) and is incremented by S (the incrementor value) until the counter is greater than B (the ending value).
- Counter is a variable.
- A, B, and S may be constants, variables, or expressions.

Flowchart Symbols (continued)

On-Page Connector



Off-Page Connector

- Flowchart sections can be connected with two different symbols.
- The on-page connector connects sections on the same page.
- The off-page connector connects flowcharts from page to page.
- The on-page connector uses letters inside the circle to indicate where the adjoining connector is located. An A connects to an A, a B to a B, etc.
- The off-page connectors uses the page number where the next part or the previous part of the flowchart is located. This allows the reader to follow the flowchart easily.
- On- and off-page connectors will have either an entrance or an exit.

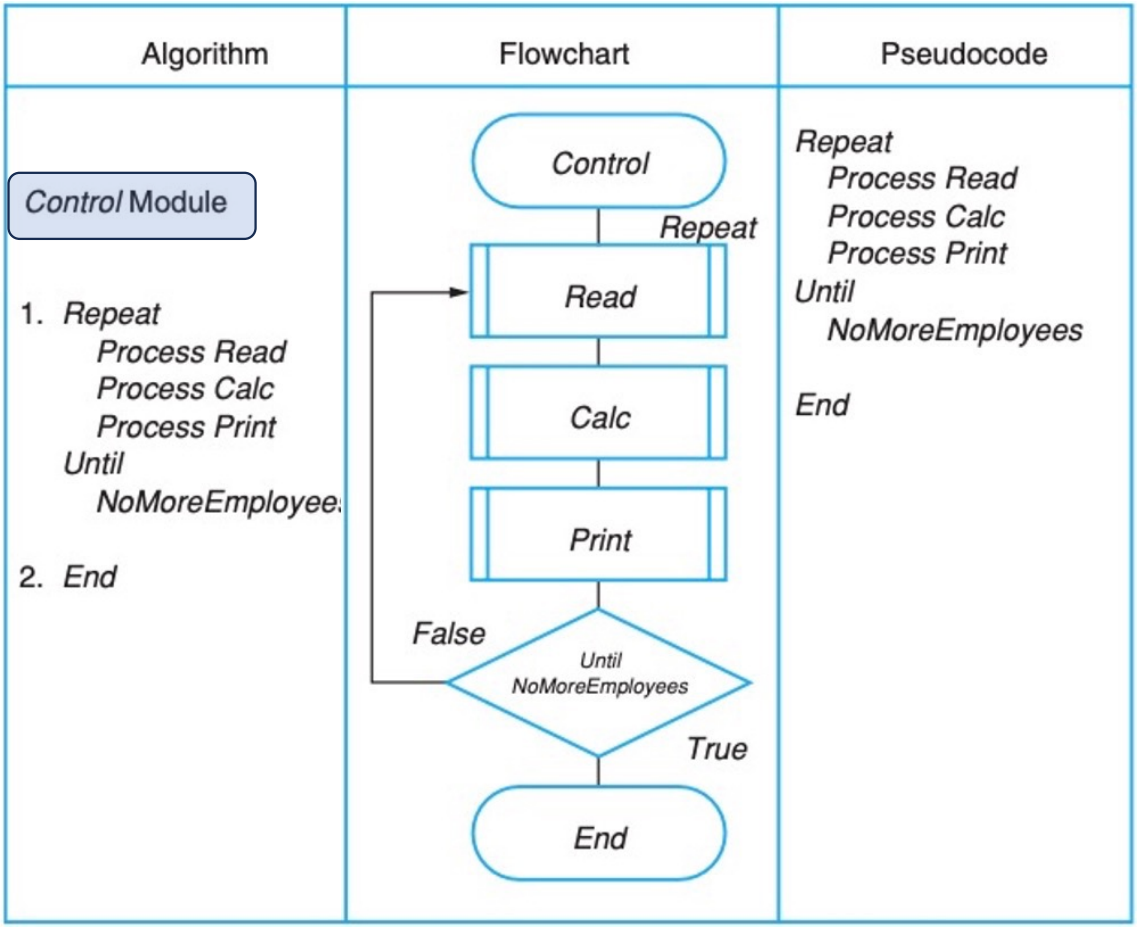
- Note that these connectors should be used as little as possible. They should be used to enhance readability. Overuse decreases readability and produces a cluttered effect.

Rules for Drawing Flowcharts

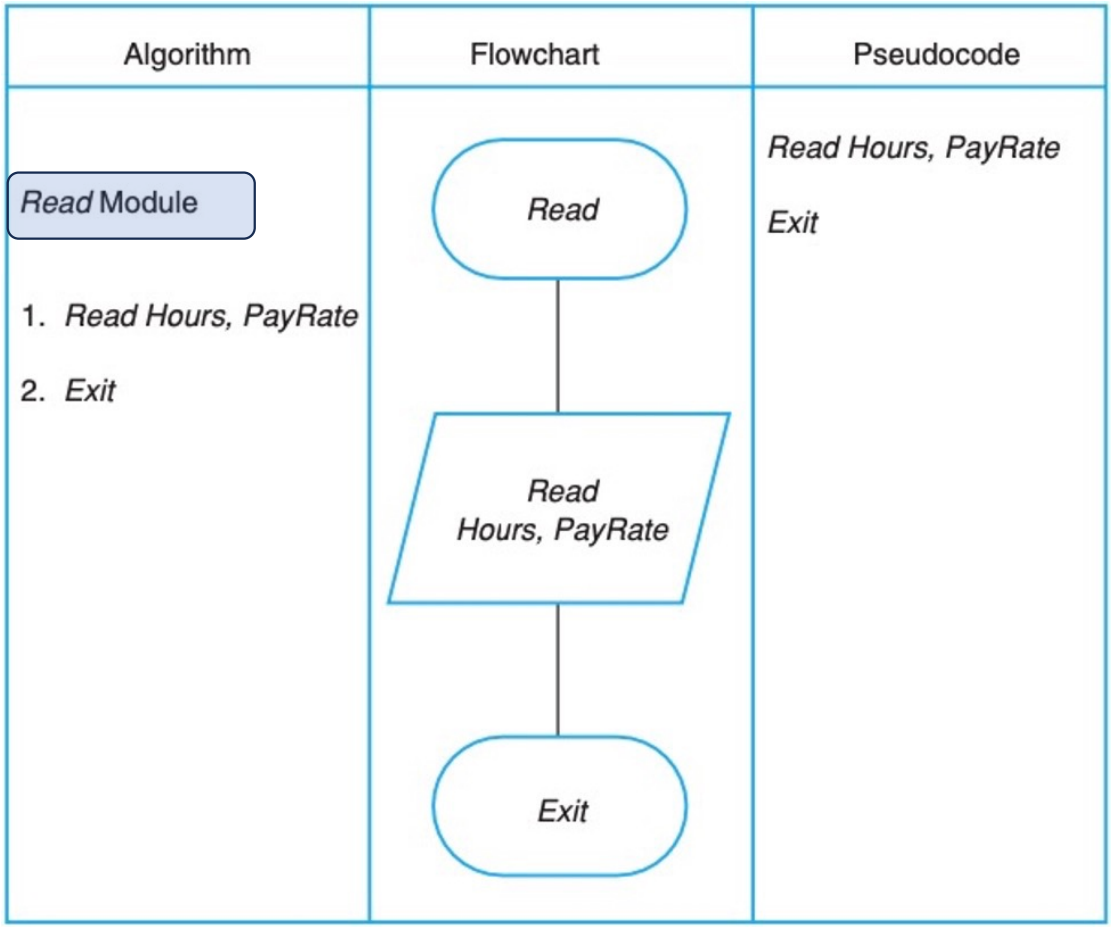
- Write the instructions inside the blocks.
- A flowchart always starts at the top of the page and flows to the bottom.
- If more than one page is needed for a flowchart, start another column on the same page or go on to another page.
- On- and off-page connectors are used to connect parts of the flowchart.
- A flowchart should not flow up, or sideways, or all over the page.
- Make the blocks big enough to write instructions so they can be easily read.
- Put the module number and name from the interactivity chart in the upper right-hand corner of the page for quick reference to the correct module.

Example

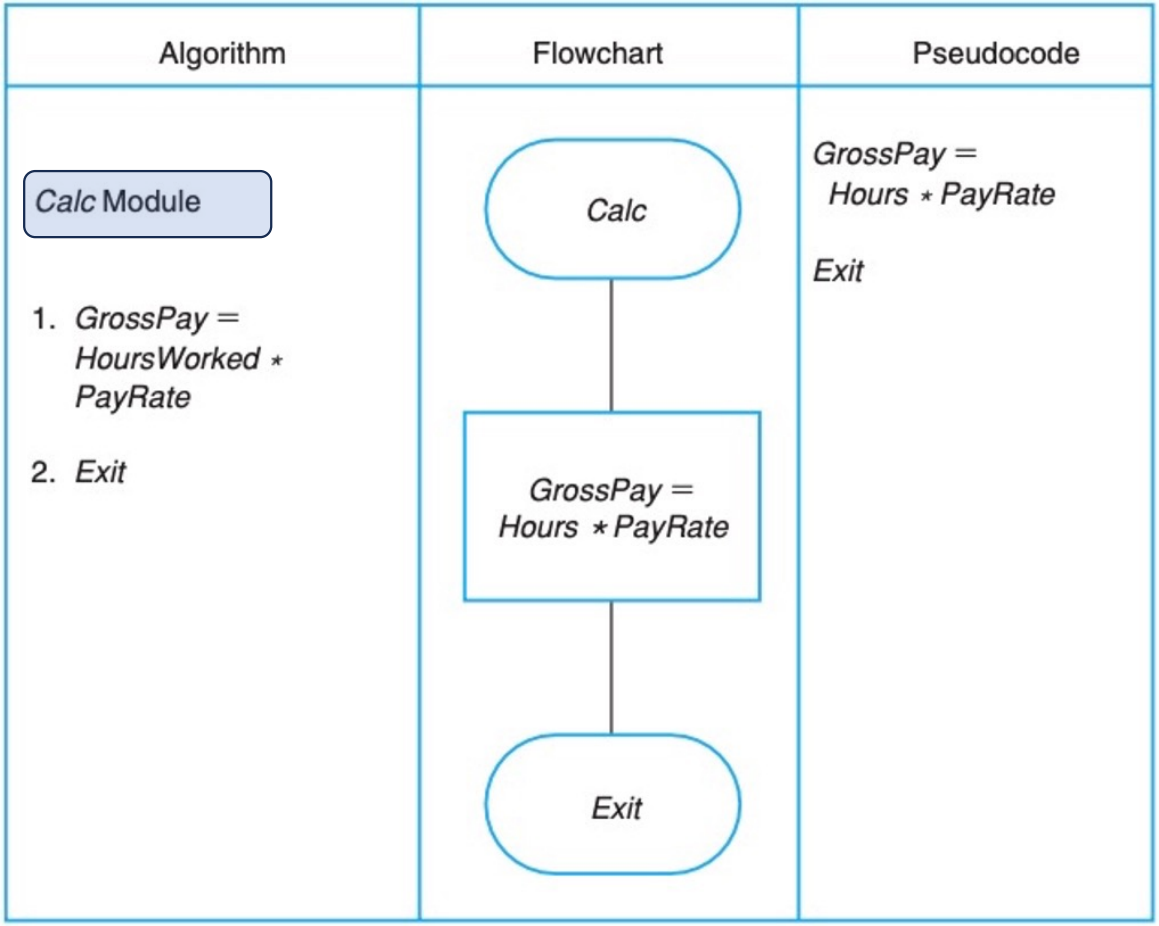
The complete flowchart for the Payroll Problem.



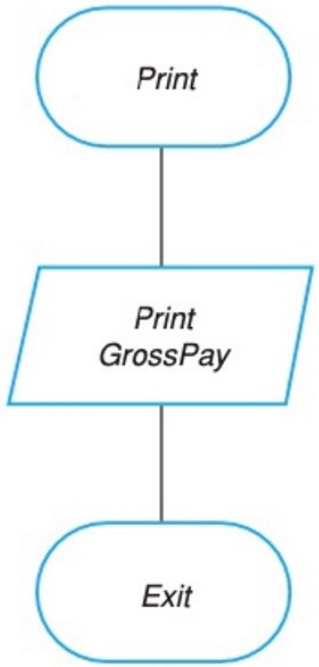
Example (continued)



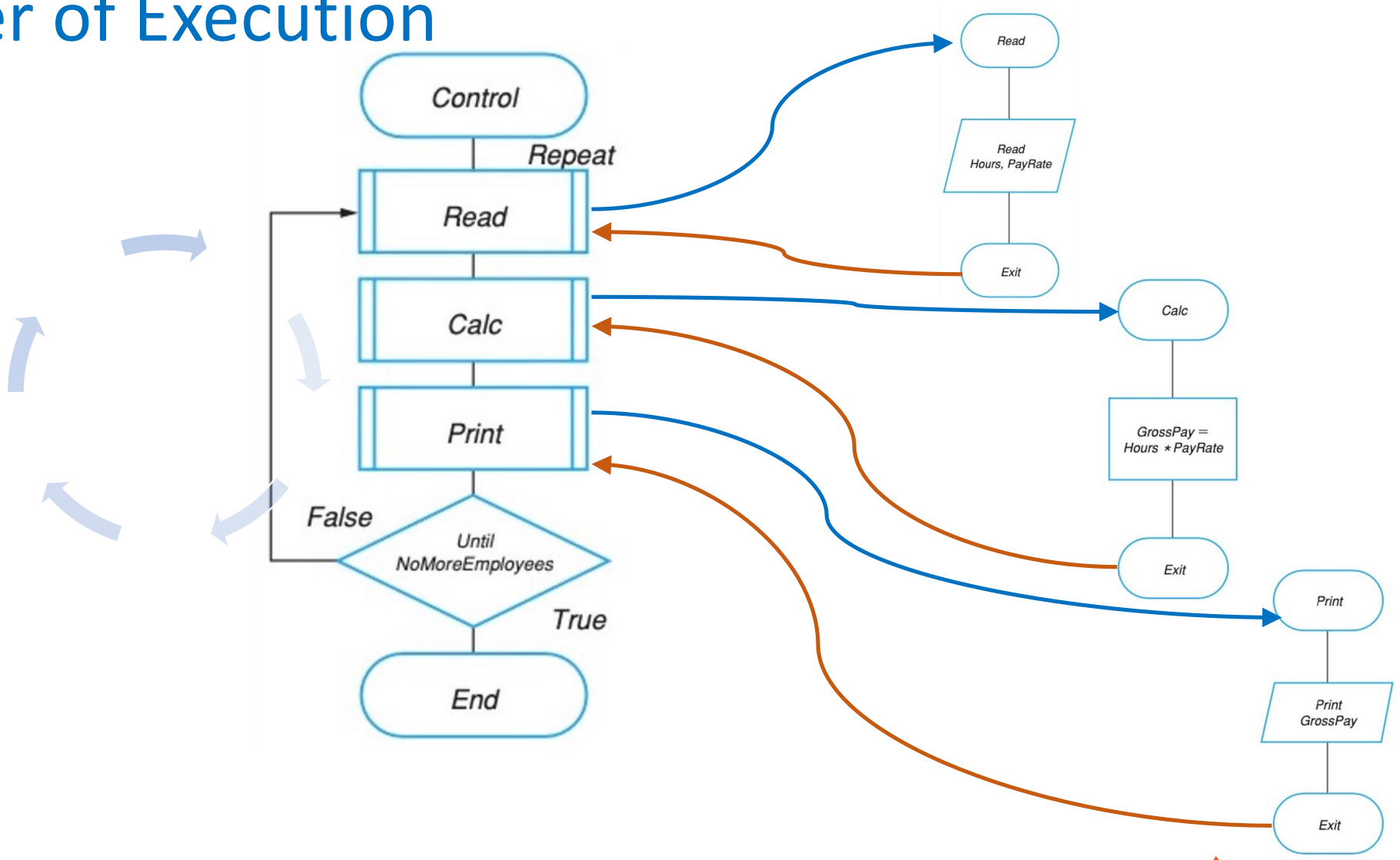
Example (continued)



Example (continued)

Algorithm	Flowchart	Pseudocode
Print Module 1. *Print Pay* 2. *Exit*		*Print Pay* *Exit*

Order of Execution



Testing The Solution

- When a solution is complete, it must be tested to ensure it meets the requirements and checked for logic errors.
- To test a solution, select a set of values for the input data and work them through every step in the solution to see whether the result is correct or not.
  - Be sure you do every step the computer would do (never assume that a step is correct).
  - If the result differs from the expected one (a bug is detected), you need to review and modify the solution to correct the instruction that caused the difference.
  - Then the testing process must be started again at the beginning.

Testing The Solution (continued)

- It is important to select test data capable of checking logic and calculations thoroughly so the correctness of the results can be checked with as much accuracy as possible.
- Logic errors, such as incorrect loop conditions or wrong conditions in a decision statement, are the hardest to find and correct.
- Correcting the solution in the problem-solving stages of program development is far easier than after the program is coded.

Coding the Solution

- The next step is writing the solution using a computer language – that is, **coding** the solution.
- Most bugs at this stage will be syntax errors.
- If the steps in developing the solution have been properly applied coding should go quickly.
- Often, the process will appear shorter and easier if you skip problem-solving steps and code directly into the computer, but this approach is almost always more time-consuming, and the resulting program is less readable and less efficient.

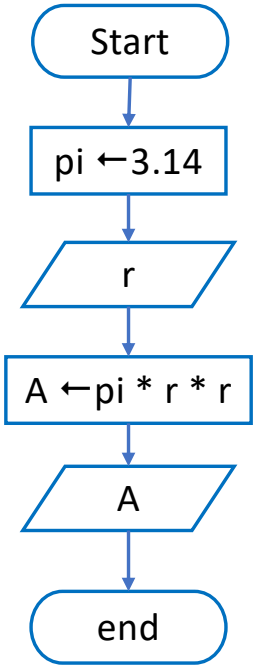
5. Documenting The Solution

- Good programming practice requires a program to be easily read by another programmer; hence, documenting a program is very important.
- Internal documentation (written as the program is coded) consists of remarks written with instructions to explain what is being done in the program.
- Proper internal documentation ensures that another programmer can read and understand the program in the least amount of time.

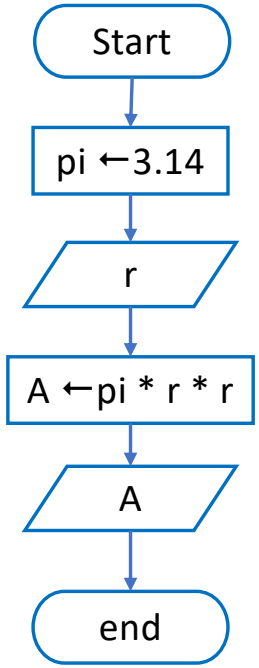
5. Documenting The Solution (continued)

- External documentation is for the end-users of the program.
- External documentation comprises the manuals or help menus.
- Proper external documentation ensures the end-user has good instructions to use this program.
- External documentation should include tutorials, input definitions, usage instructions, installation instructions, and command explanations.

Exercise 1: Write the pseudocode of the following flowchart.

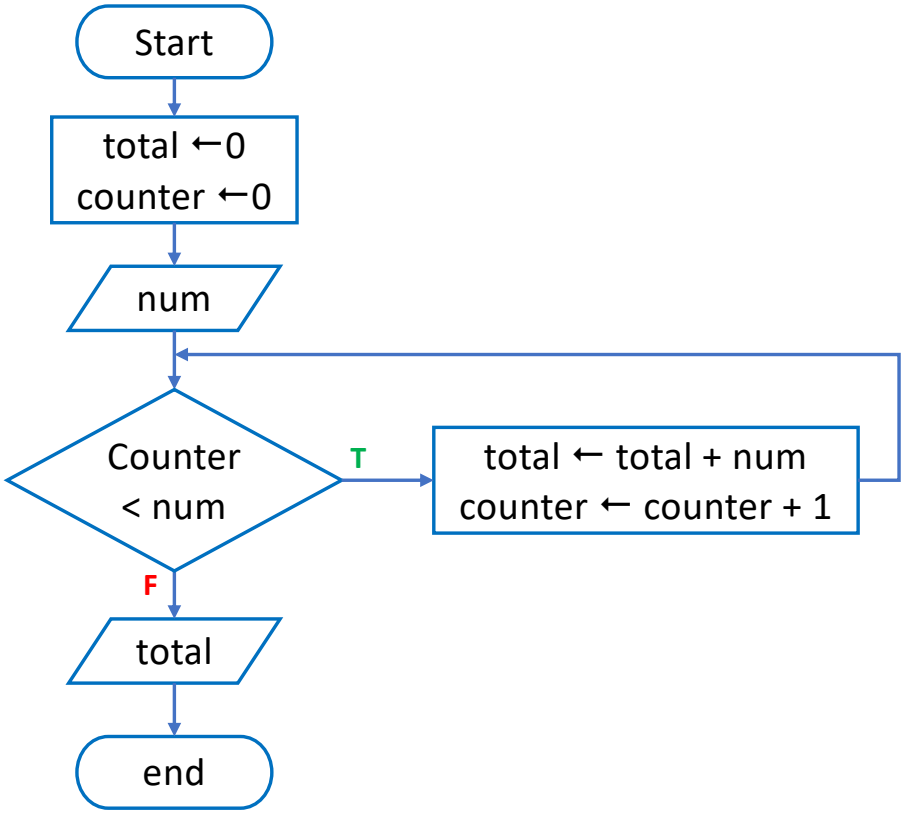


Exercise 1: Write the pseudocode of the following flowchart.

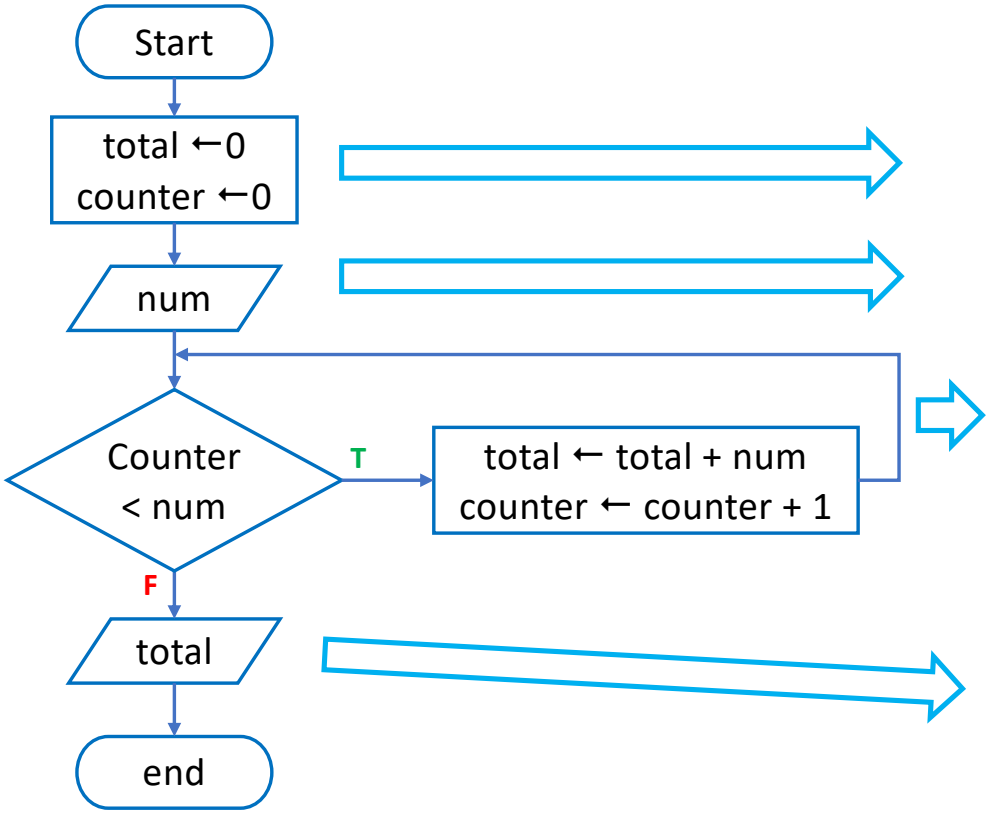



```
set 3.14 to pi

input r from the user

calculate pi*r*r and set the result to A

display A
```

Exercise 2: Write the pseudocode of the following flowchart.



Exercise 2: Write the pseudocode of the following flowchart.



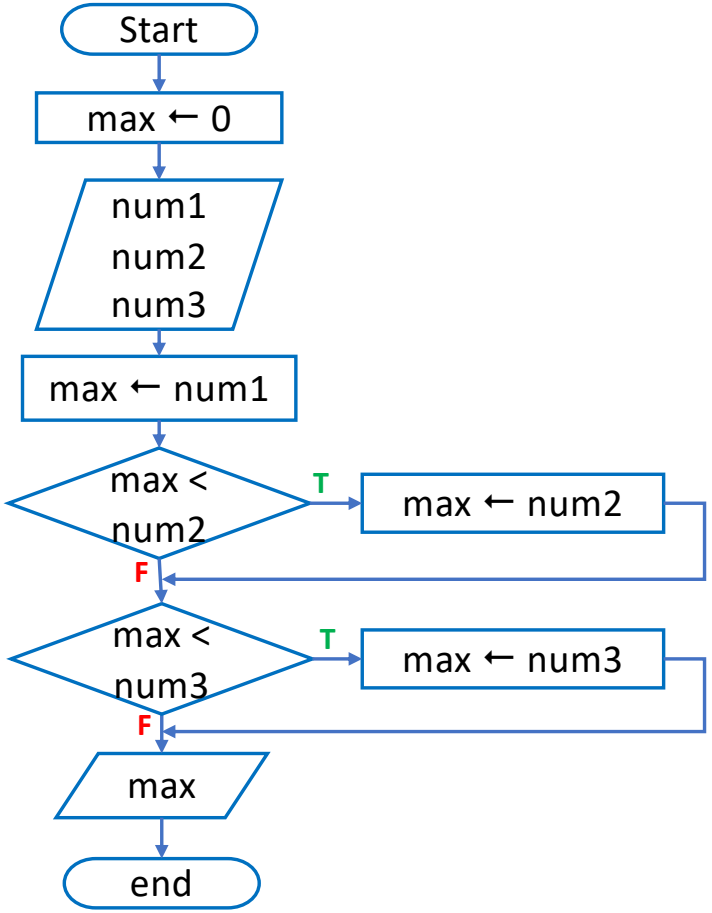

```
set 0 to total
set 0 to counter
read num
WHILE counter is less then num
        add num to total
        increment counter by one
END WHILE
display total
```

Exercise 3: Draw the flowchart of the following pseudocode.

```
set 0 to max
read num1, num2, and num3
set max to num1
IF max is less then num2 THEN
      set num2 to max
END IF
IF max is less then num3 THEN
      set num3 to max
END IF
display max
```

Exercise 3: Draw the flowchart of the following pseudocode.

```
set 0 to max
read num1, num2, and num3
set max to num1
IF max is less then num2 THEN
      set num2 to max
END IF
IF max is less then num3 THEN
      set num3 to max
END IF
display max
```



Problem Set 02

Draw the flowchart and write the pseudocode for the following:

P1. Display the multiplication of numbers from 15 to 250.

P2. Get six numbers from the user and display their average.

P3. Ask the user to enter a number from the keyboard and then display all the even numbers up to that number.

P4. Display all the integer numbers from 1 to 30 divisible by 3.

P5. Get two integer numbers from the user and display all the integer numbers between them, which are divisible by 7.

References

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