

ALGORITHMS AND FLOWCHARTS

A typical programming task can be divided into two phases:

- ***Problem solving phase***

- produce an ordered sequence of steps that describe solution of problem
- this sequence of steps is called an ***algorithm***

- ***Implementation phase***

- implement the program in some programming language

Steps in Problem Solving

- First produce a general algorithm (one can use *pseudocode*)
- Refine the algorithm successively to get step by step detailed *algorithm* that is very close to a computer language.
- *Pseudocode* is an artificial and informal language that helps programmers develop algorithms. Pseudocode is very similar to everyday English.

The Flowchart

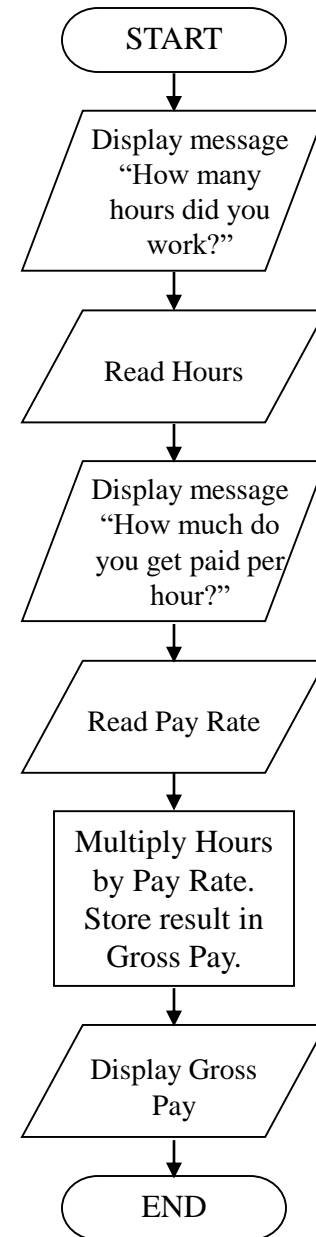
Program flowcharts show the sequence of instructions in a single program or subroutine. Different symbols are used to draw each type of flowchart.

A Flowchart

- shows logic of an algorithm
- emphasizes individual steps and their interconnections
- e.g. control flow from one action to the next

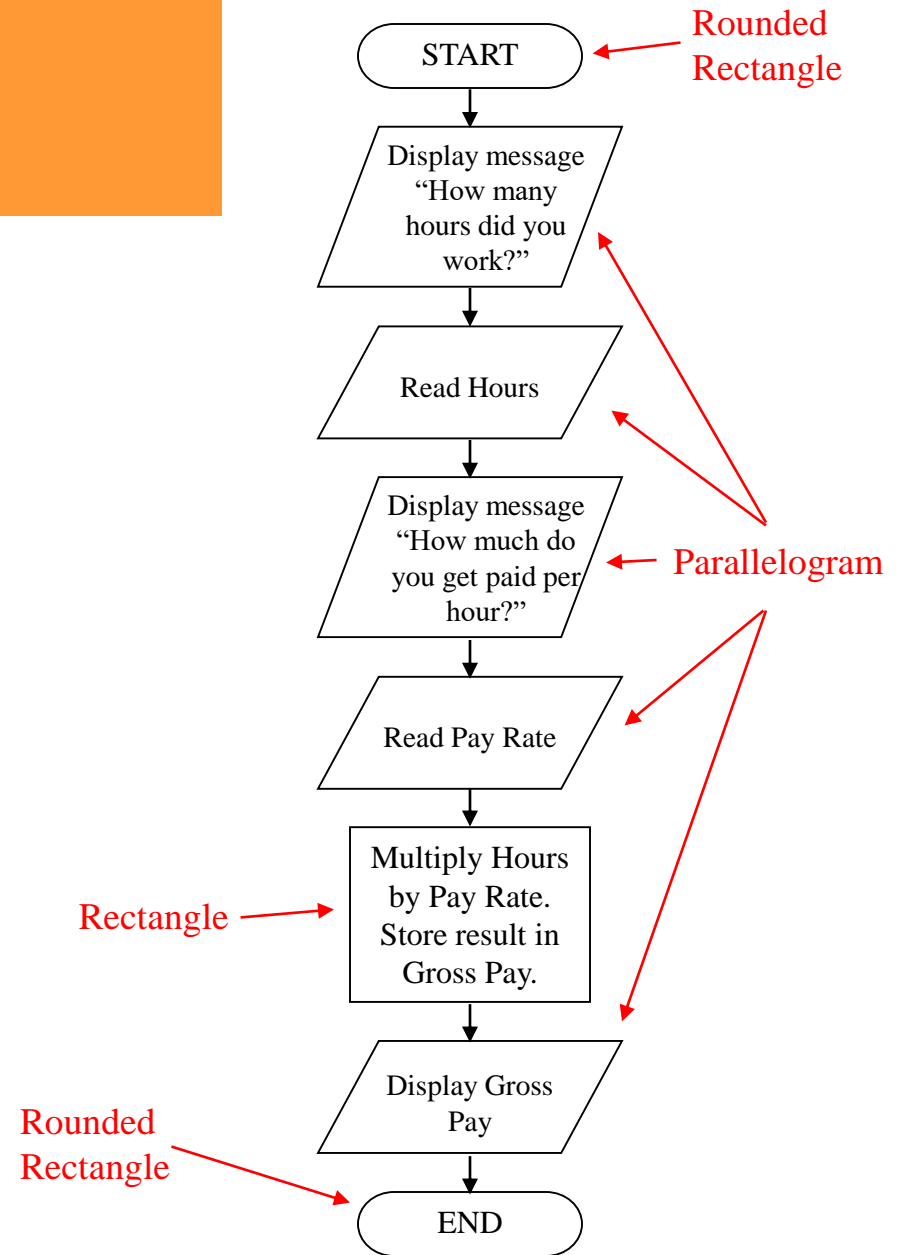
What is a Flowchart?

- A flowchart is a diagram that depicts the “flow” of a program.
- The figure shown here is a flowchart for pay-calculating program.



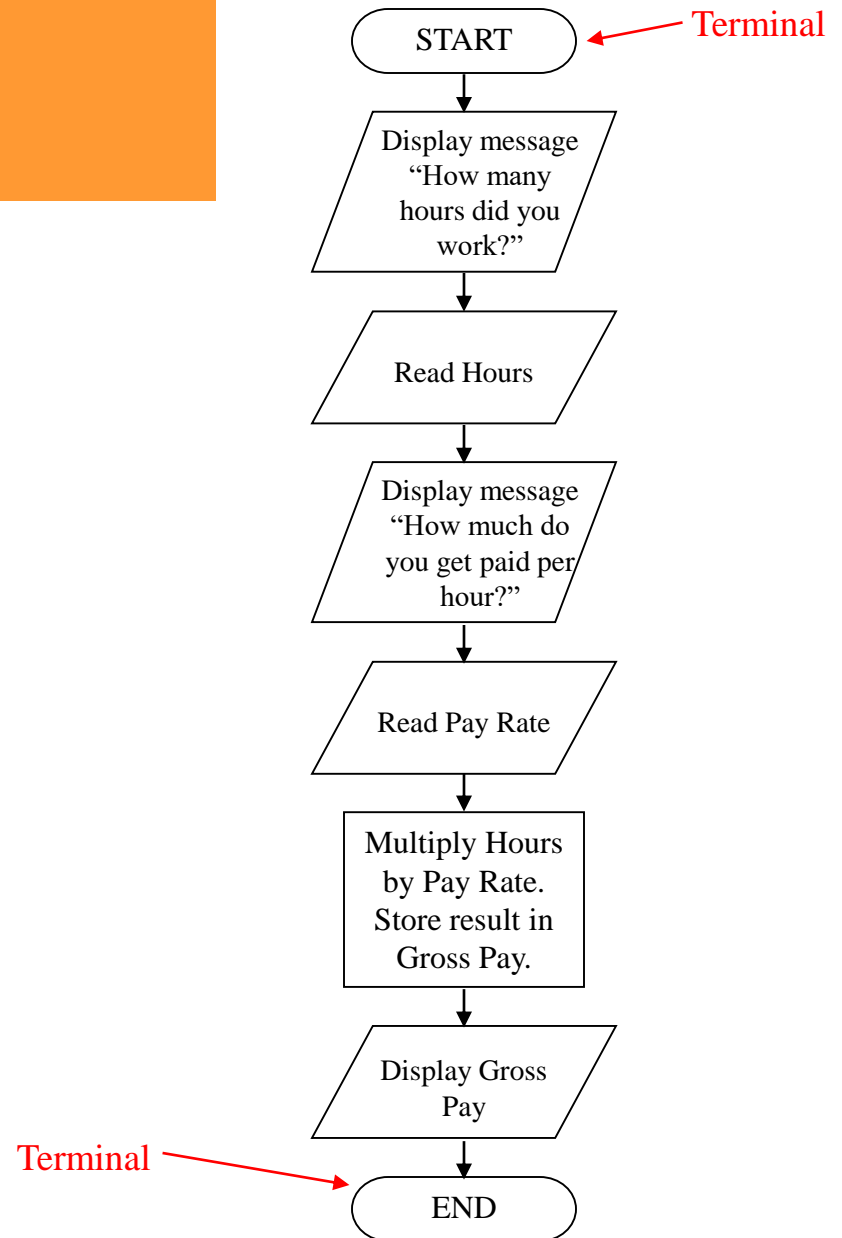
Basic Flowchart Symbols

- Notice there are three types of symbols in this flowchart:
 - rounded rectangles
 - parallelograms
 - a rectangle
- Each symbol represents a different type of operation.



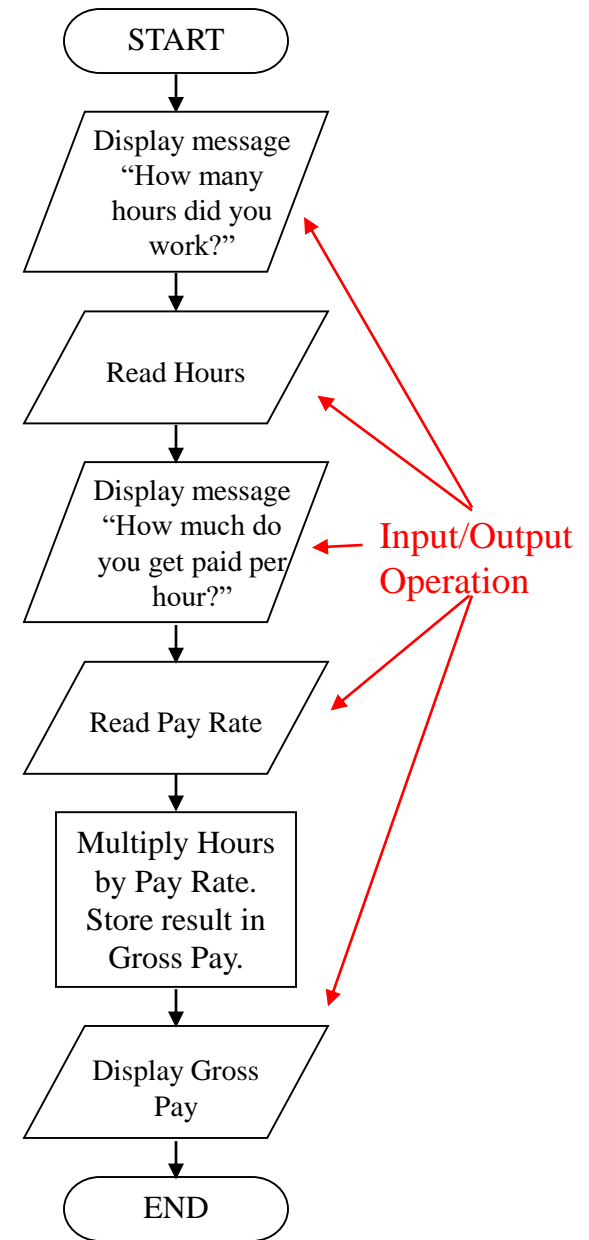
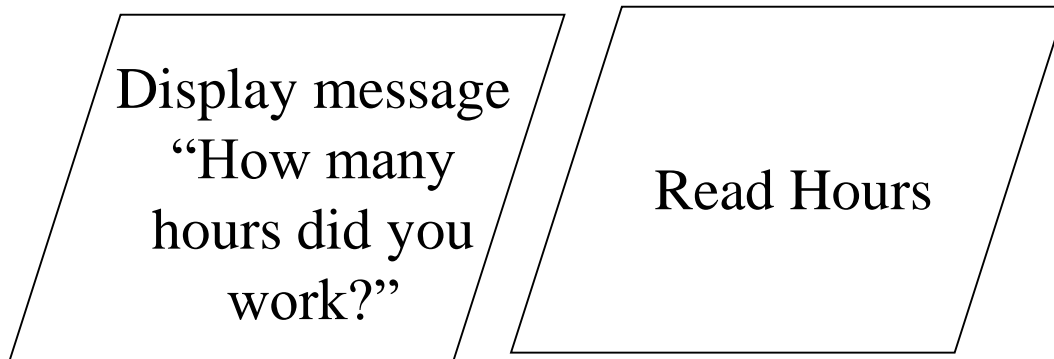
Basic Flowchart Symbols

- Terminals
 - represented by rounded rectangles
 - indicate a starting or ending point



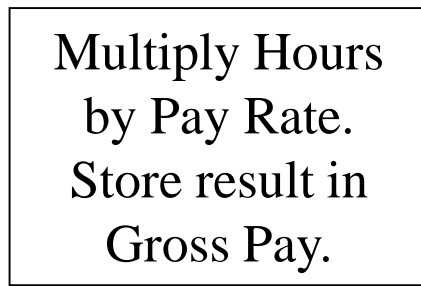
Basic Flowchart Symbols

- Input/Output Operations
 - represented by parallelograms
 - indicate an input or output operation

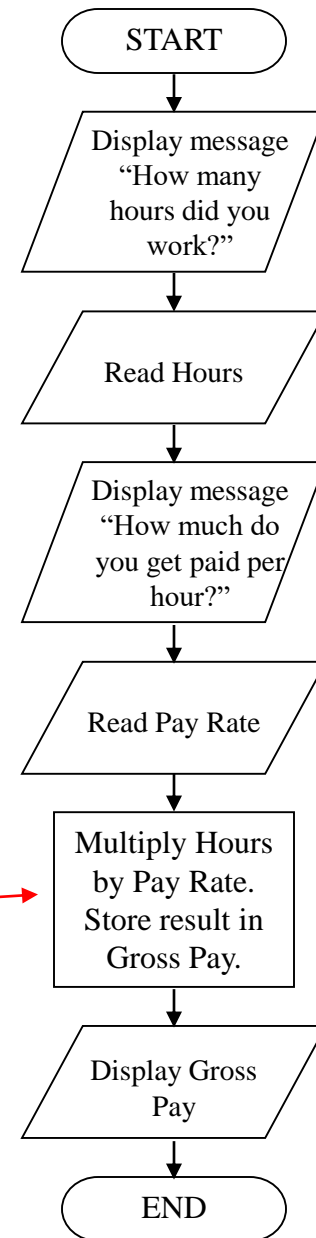


Basic Flowchart Symbols

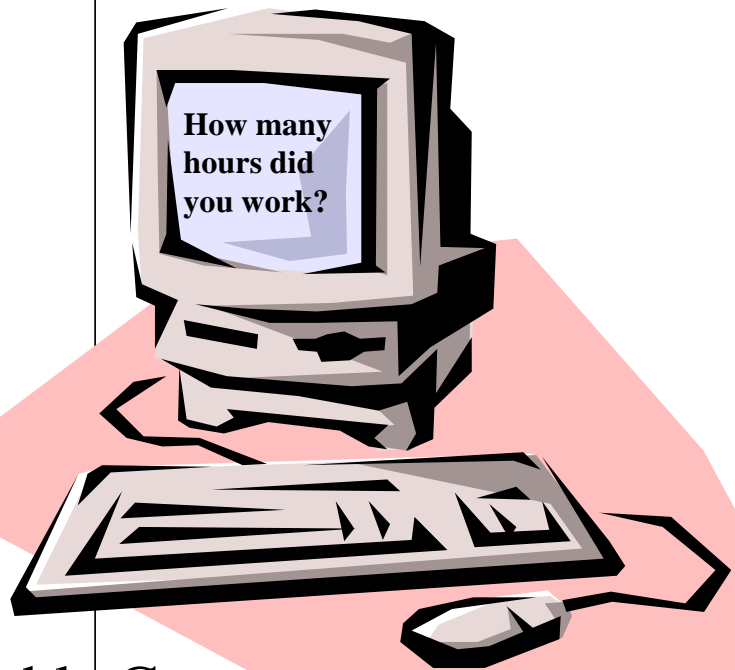
- Processes
 - represented by rectangles
 - indicates a process such as a mathematical computation or variable assignment



Process →



Stepping Through the Flowchart

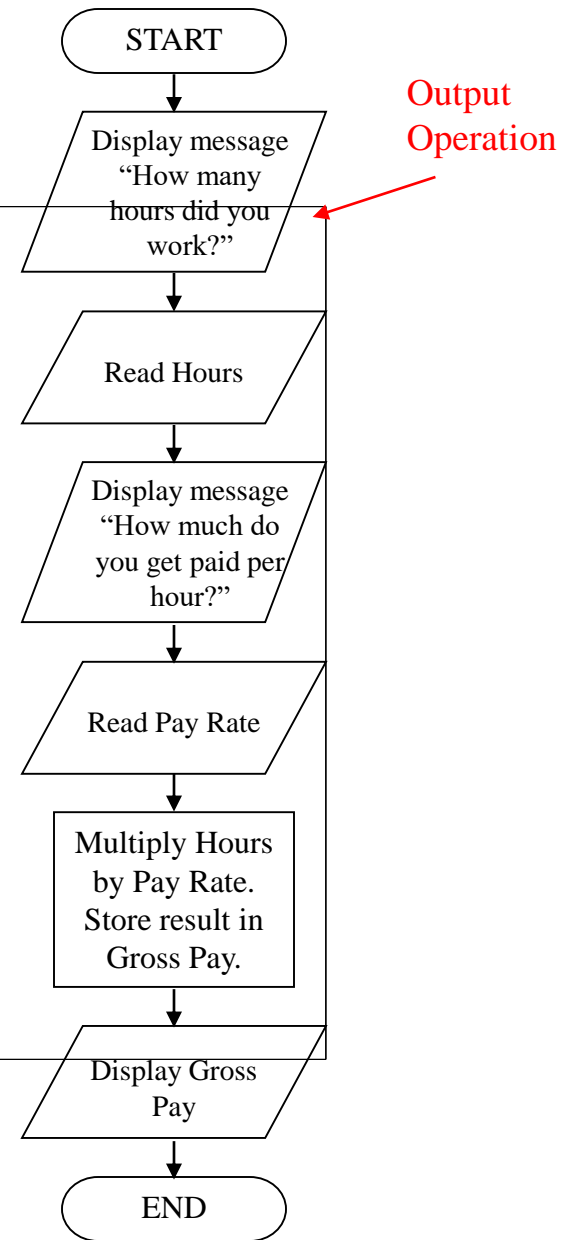


Variable Contents:

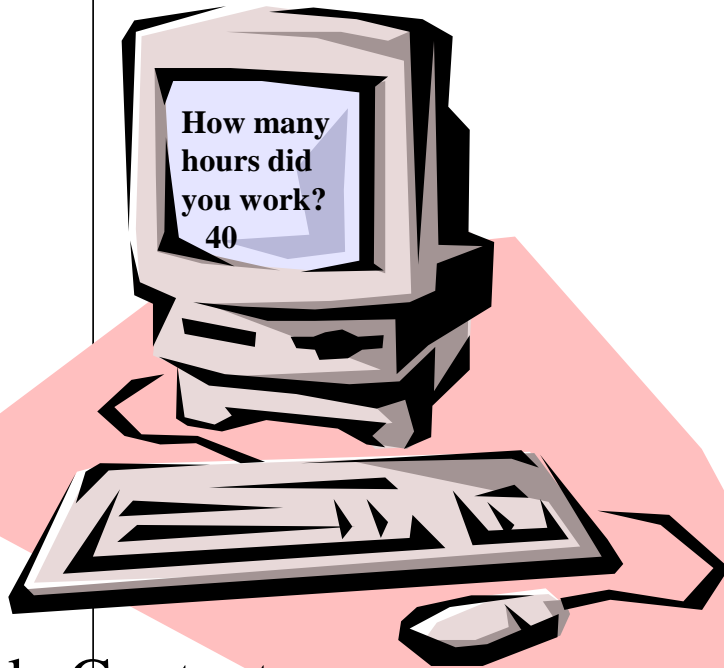
Hours: ?

Pay Rate: ?

Gross Pay: ?



Stepping Through the Flowchart



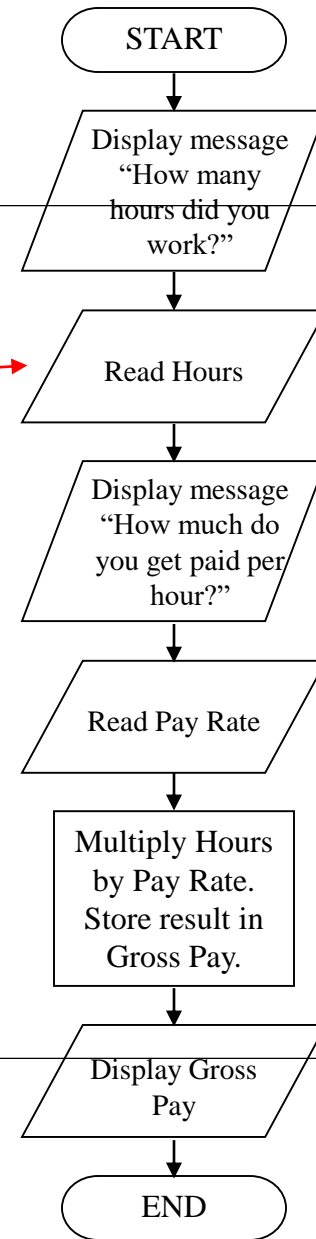
Variable Contents:

Hours: 40

Pay Rate: ?

Gross Pay: ?

Input
Operation
(User types
40)



Stepping Through the Flowchart



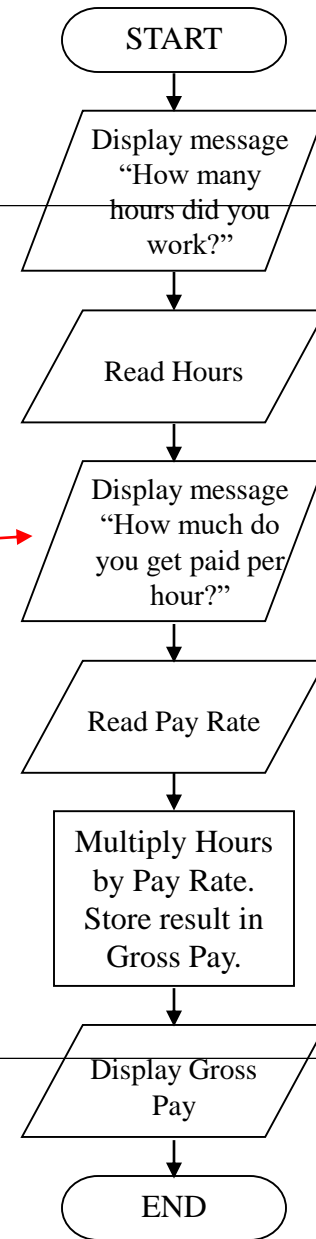
Variable Contents:

Hours: 40

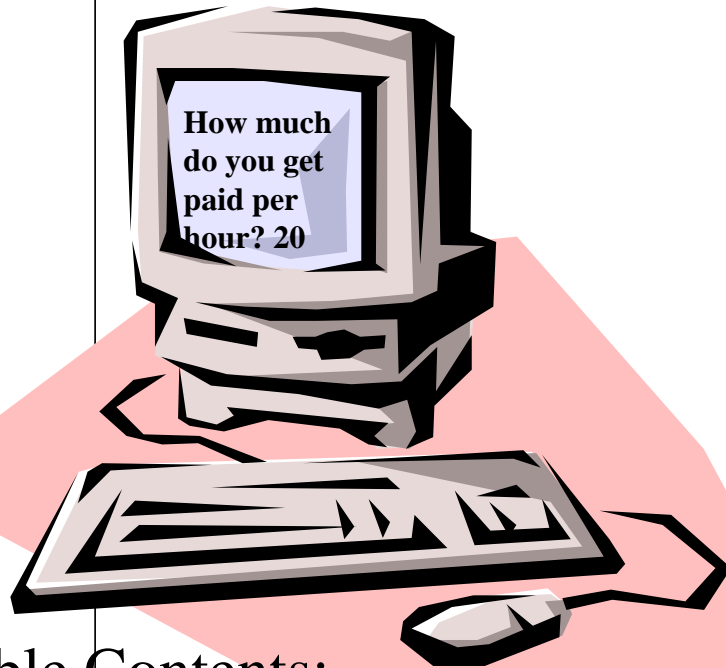
Pay Rate: ?

Gross Pay: ?

Output
Operation



Stepping Through the Flowchart



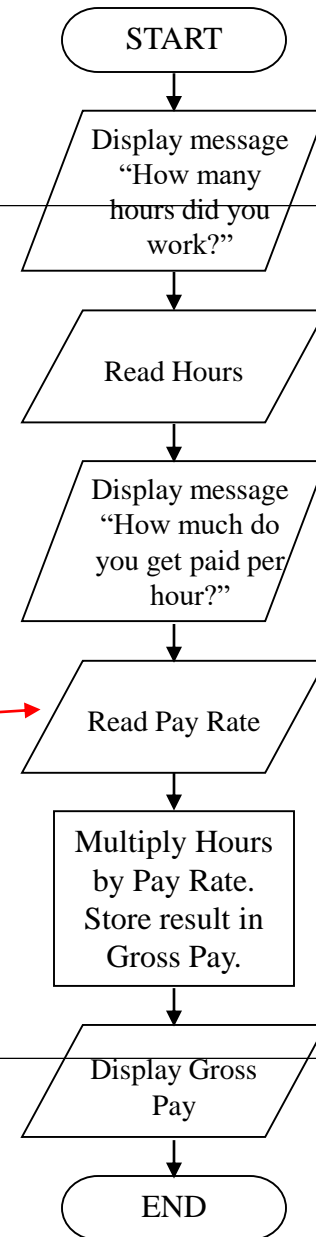
Variable Contents:

Hours: 40

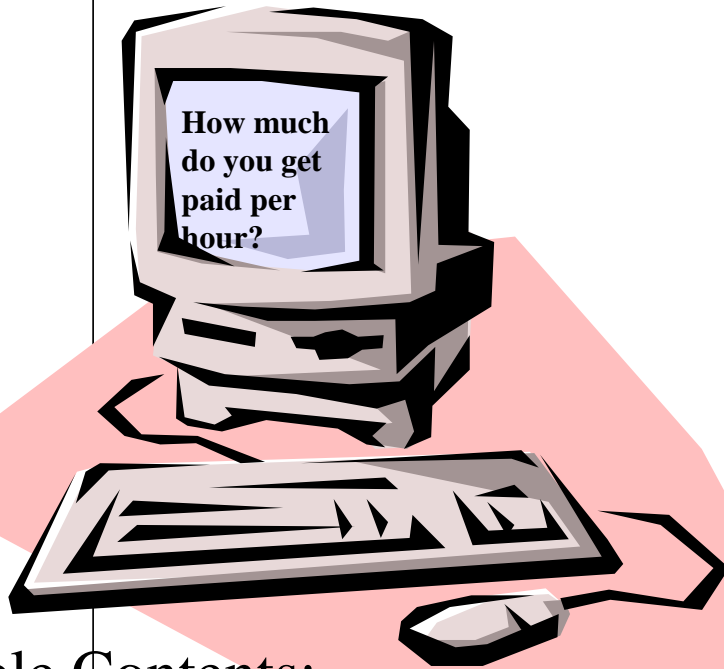
Pay Rate: 20

Gross Pay: ?

Input
Operation
(User types
20)



Stepping Through the Flowchart



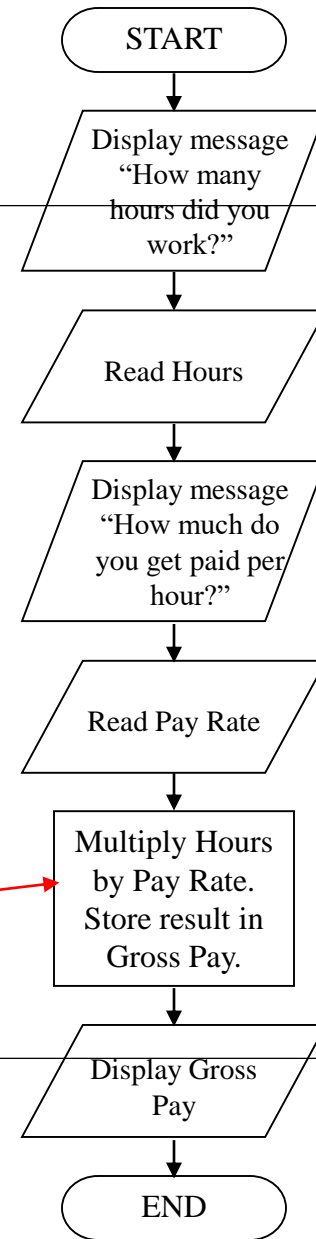
Variable Contents:

Hours: 40

Pay Rate: 20

Gross Pay: 800

Process: The product of 40 times 20 is stored in Gross Pay



Stepping Through the Flowchart



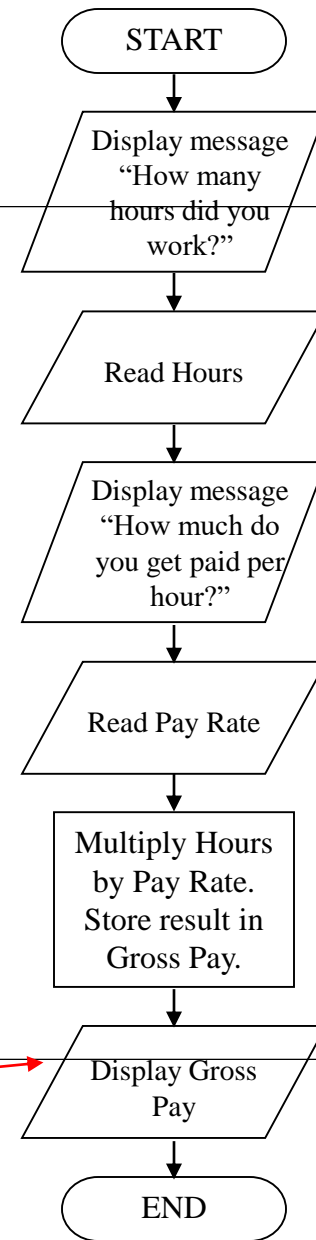
Variable Contents:

Hours: 40

Pay Rate: 20

Gross Pay: 800

Output
Operation

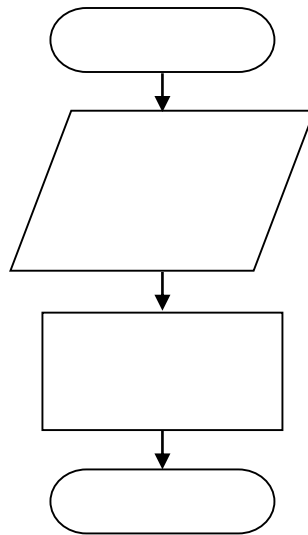


Four Flowchart Structures

- Sequence
- Decision
- Repetition
- Case

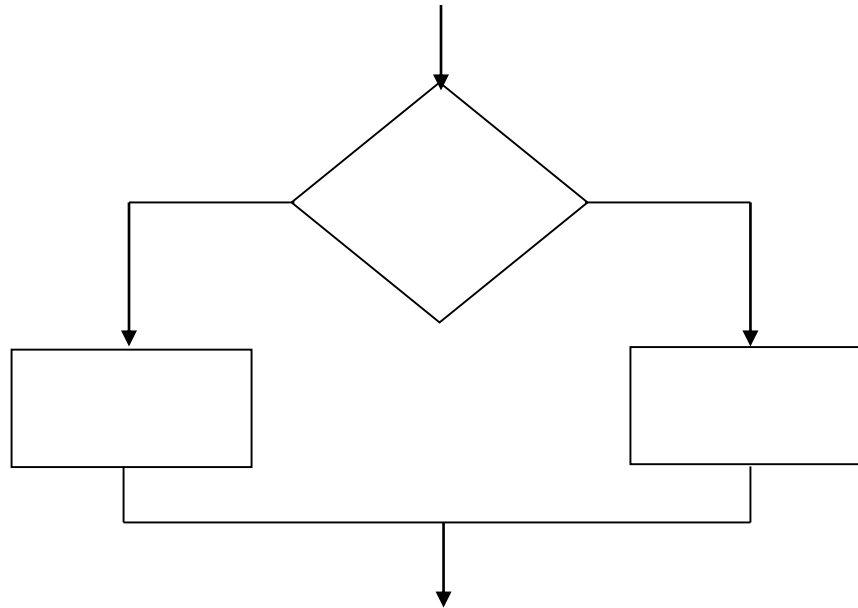
Sequence Structure

- a series of actions are performed in sequence
- The pay-calculating example was a sequence flowchart.



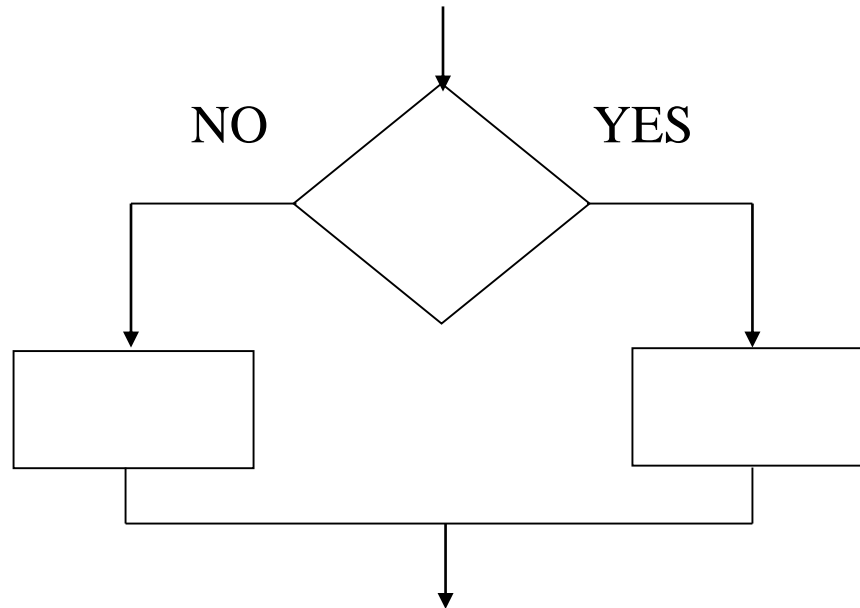
Decision Structure

- One of two possible actions is taken, depending on a condition.



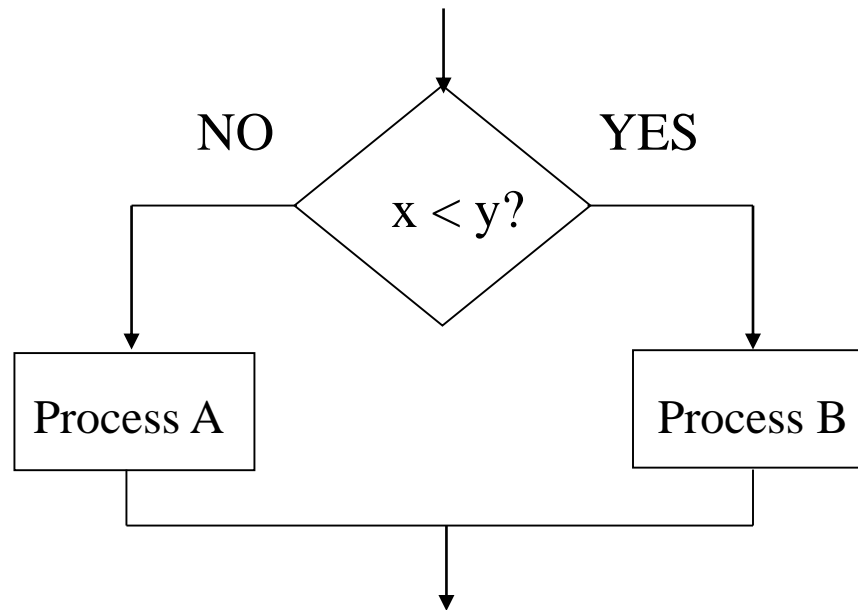
Decision Structure

- A new symbol, the diamond, indicates a yes/no question. If the answer to the question is yes, the flow follows one path. If the answer is no, the flow follows another path



Decision Structure

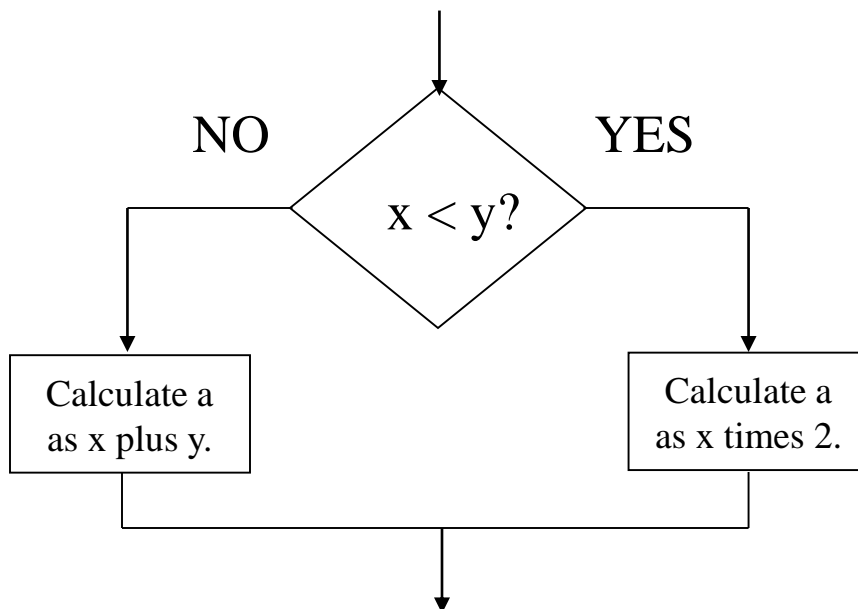
- In the flowchart segment below, the question “is $x < y$?” is asked. If the answer is no, then process A is performed. If the answer is yes, then process B is performed.



Decision Structure

- The flowchart segment below shows how a decision structure is expressed in C++ as an if/else statement.

Flowchart



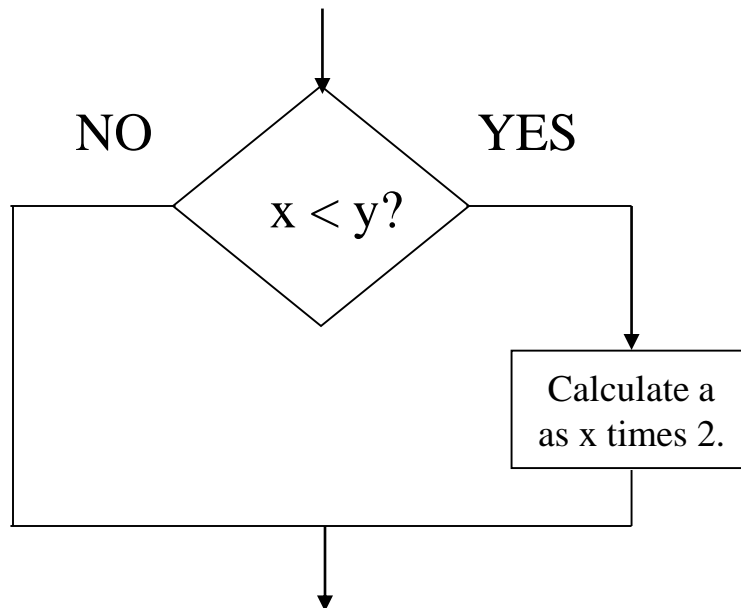
C++ Code

```
if (x < y)
    a = x * 2;
else
    a = x + y;
```

Decision Structure

- The flowchart segment below shows a decision structure with only one action to perform. It is expressed as an if statement in C++ code.

Flowchart

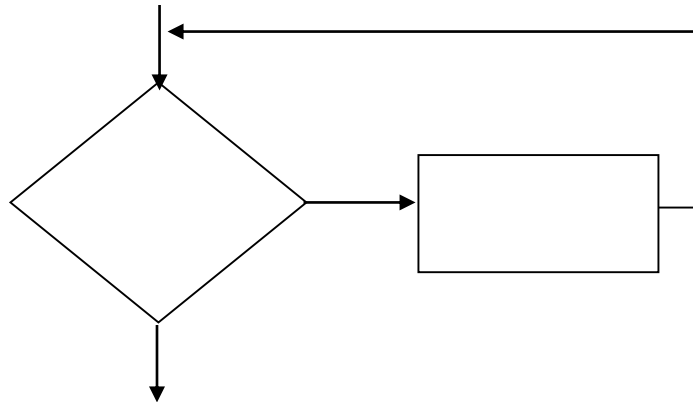


C++ Code

```
if (x < y)
    a = x * 2;
```

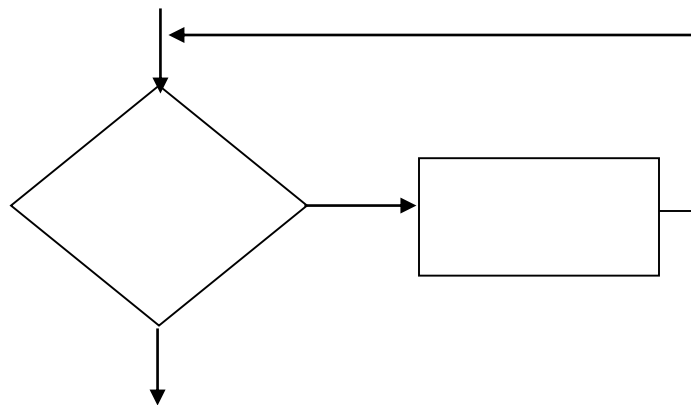
Repetition Structure

- A repetition structure represents part of the program that repeats. This type of structure is commonly known as a loop.



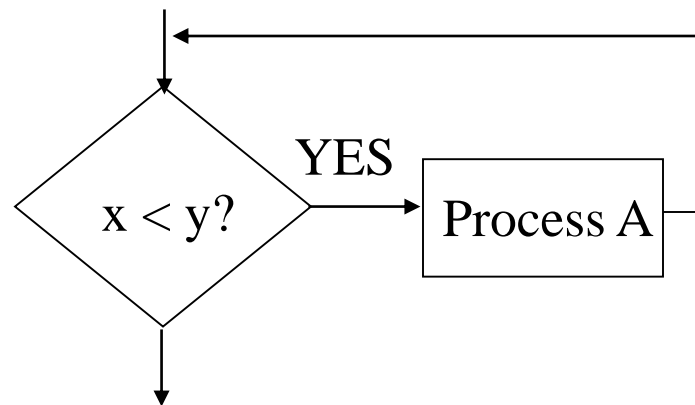
Repetition Structure

- Notice the use of the diamond symbol. A loop tests a condition, and if the condition exists, it performs an action. Then it tests the condition again. If the condition still exists, the action is repeated. This continues until the condition no longer exists.



Repetition Structure

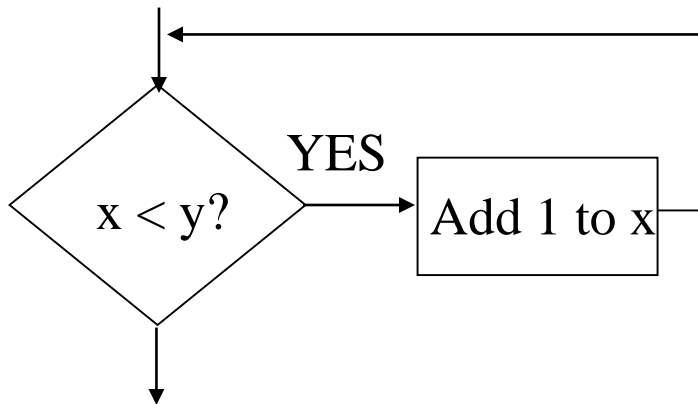
- In the flowchart segment, the question “is $x < y$?” is asked. If the answer is yes, then Process A is performed. The question “is $x < y$?” is asked again. Process A is repeated as long as x is less than y . When x is no longer less than y , the repetition stops and the structure is exited.



Repetition Structure

- The flowchart segment below shows a repetition structure expressed in C++ as a while loop.

Flowchart

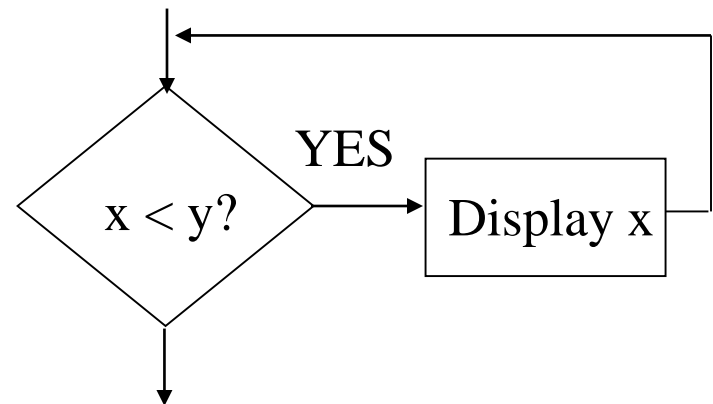


C++ Code

```
while (x < y)
    x++;
```

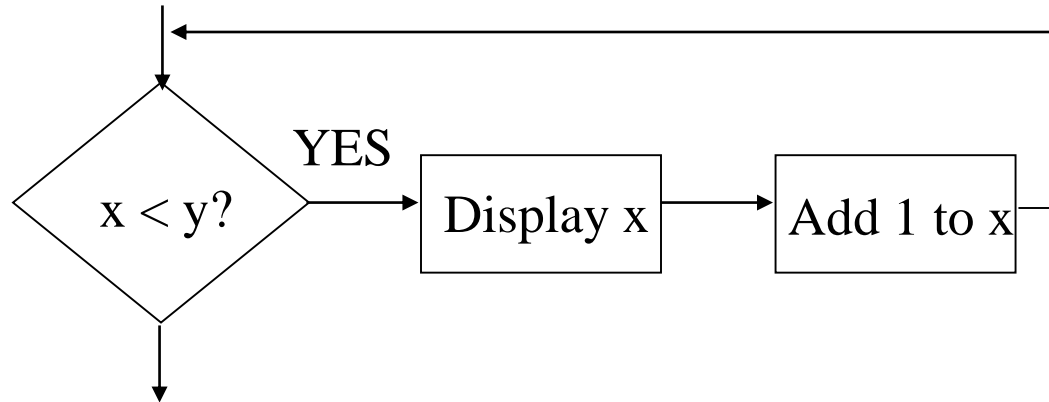
Controlling a Repetition Structure

- The action performed by a repetition structure must eventually cause the loop to terminate. Otherwise, an infinite loop is created.
- In this flowchart segment, x is never changed. Once the loop starts, it will never end.
- QUESTION: How can this flowchart be modified so it is no longer an infinite loop?



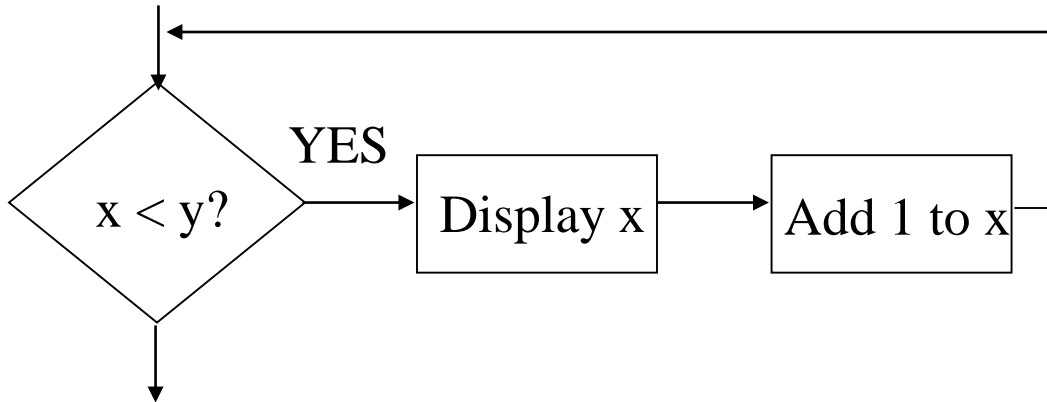
Controlling a Repetition Structure

- ANSWER: By adding an action within the repetition that changes the value of x .



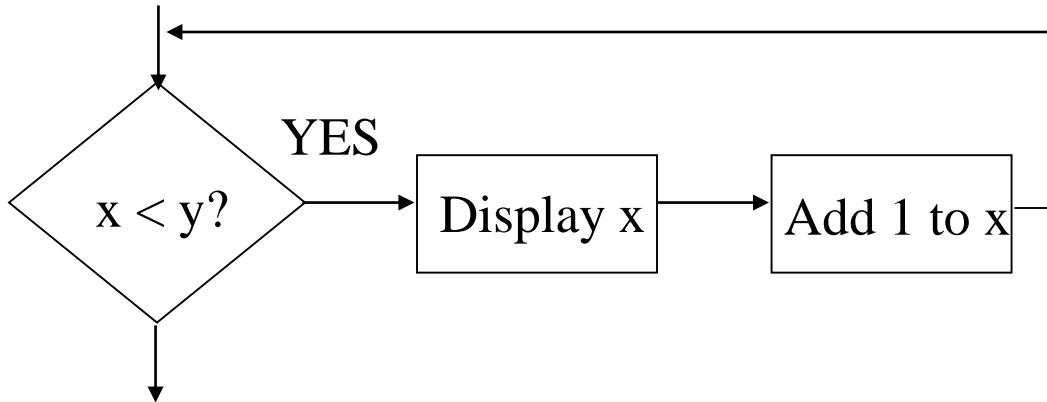
A Pre-Test Repetition Structure

- This type of structure is known as a pre-test repetition structure. The condition is tested *BEFORE* any actions are performed.



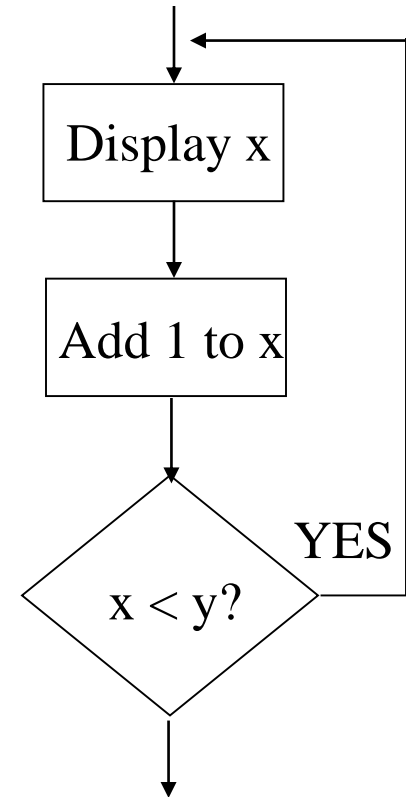
A Pre-Test Repetition Structure

- In a pre-test repetition structure, if the condition does not exist, the loop will never begin.



A Post-Test Repetition Structure

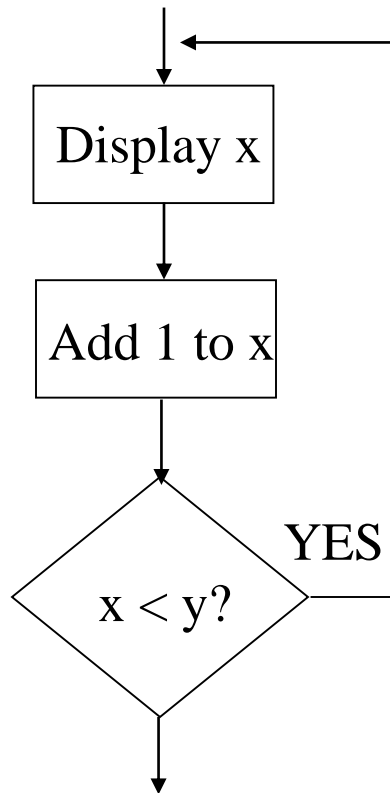
- This flowchart segment shows a post-test repetition structure.
- The condition is tested *AFTER* the actions are performed.
- A post-test repetition structure always performs its actions at least once.



A Post-Test Repetition Structure

- The flowchart segment below shows a post-test repetition structure expressed in C++ as a do-while loop.

Flowchart



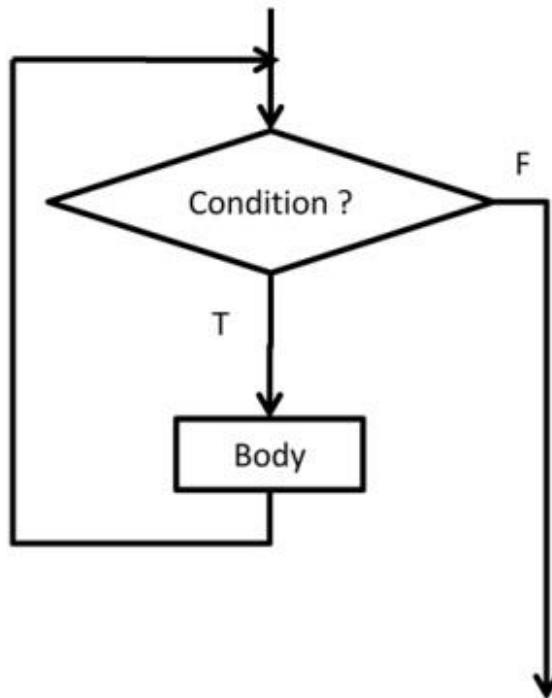
C++ Code

```
do
{
    cout << x << endl;
    x++;
} while (x < y);
```

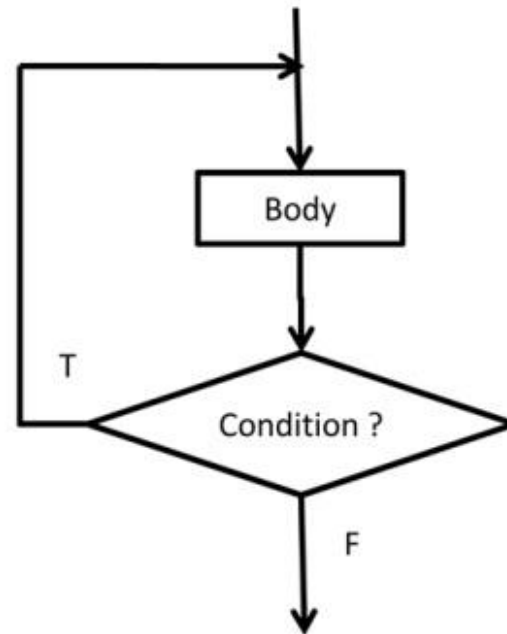
Post-Test vs Pre-Test Repetition Structure

While versus Do-While Loops

while(condition)
body;

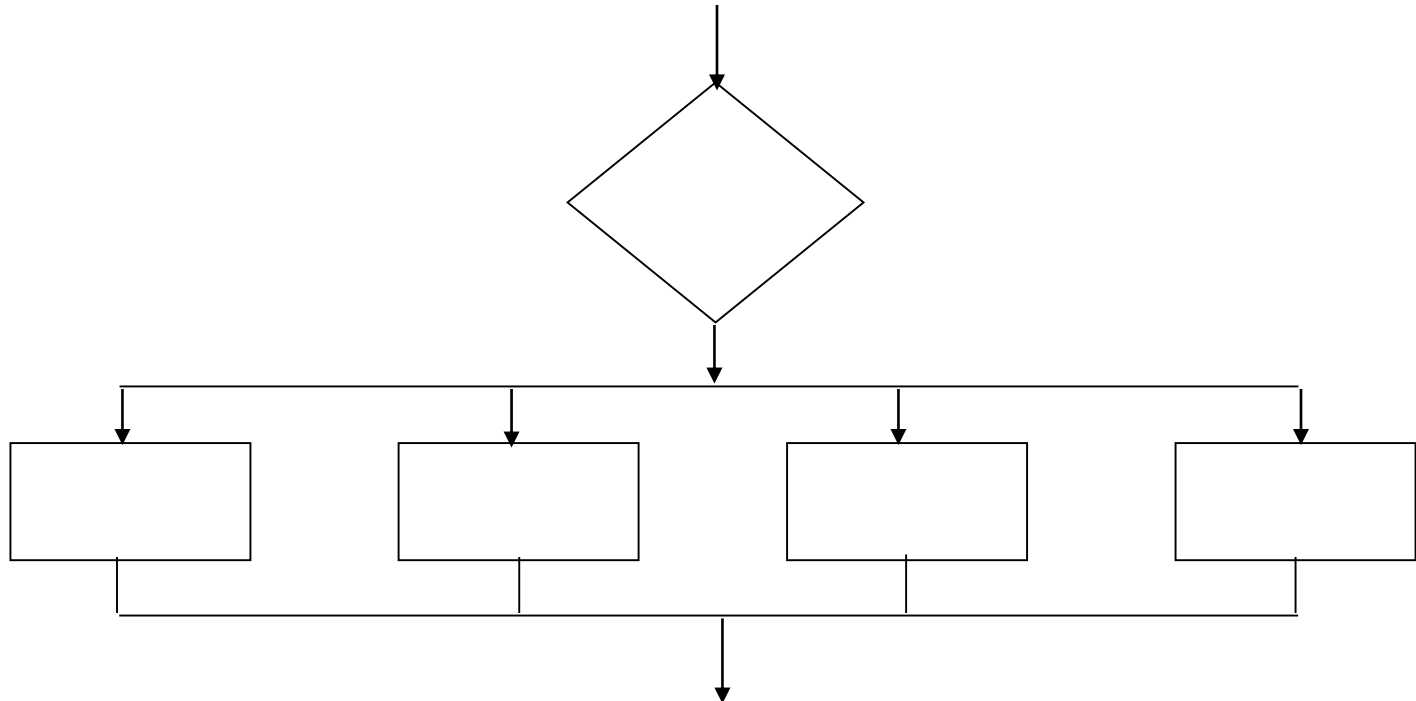


do {
body;
} while(condition);



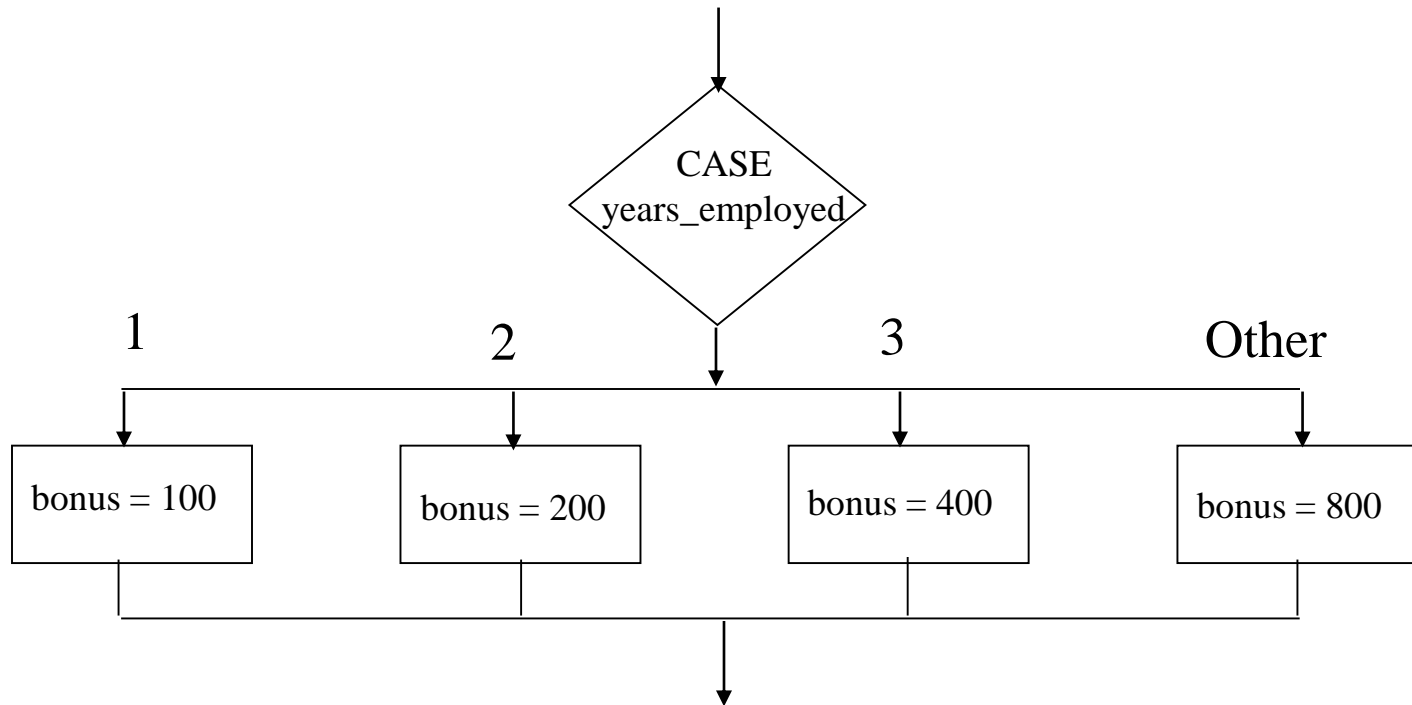
Case Structure

- One of several possible actions is taken, depending on the contents of a variable.

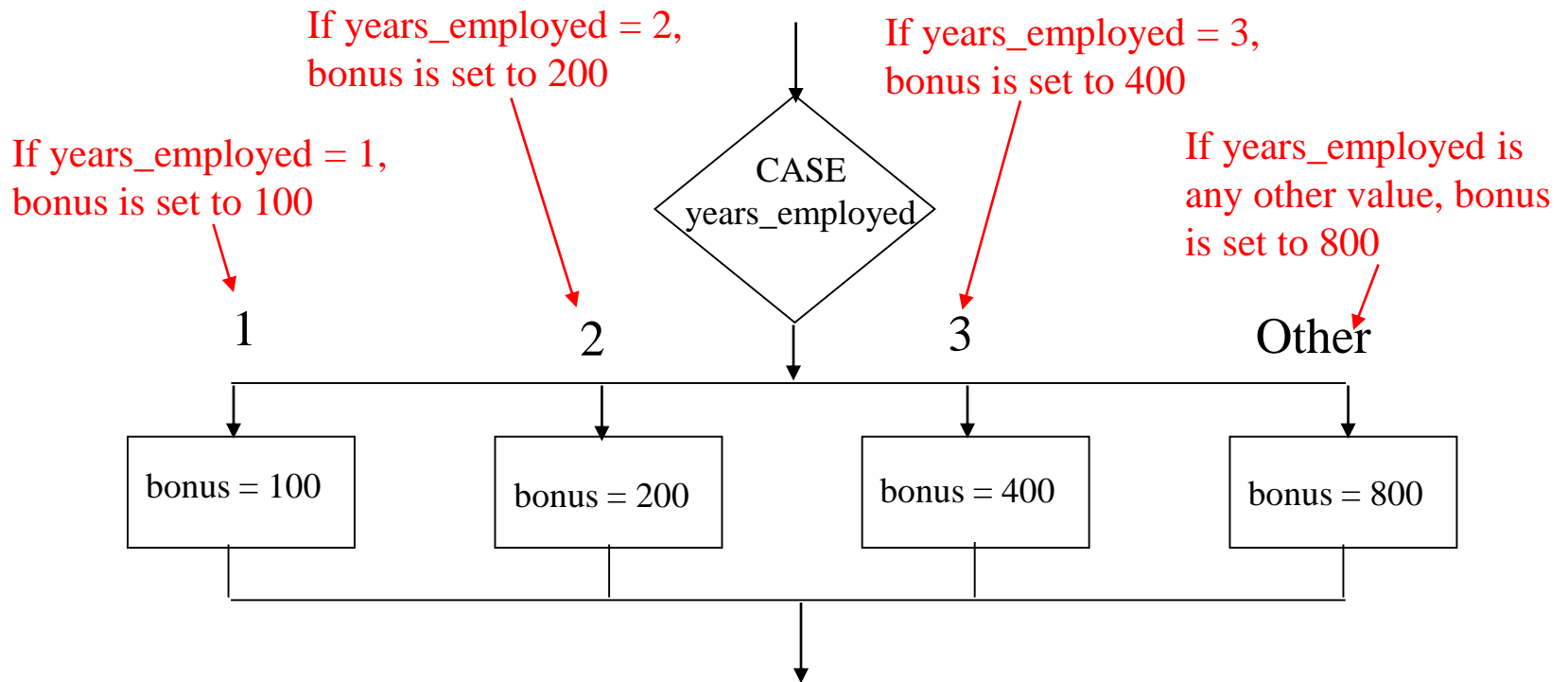


Case Structure

- The structure below indicates actions to perform depending on the value in `years_employed`.

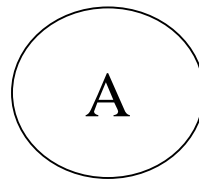


Case Structure



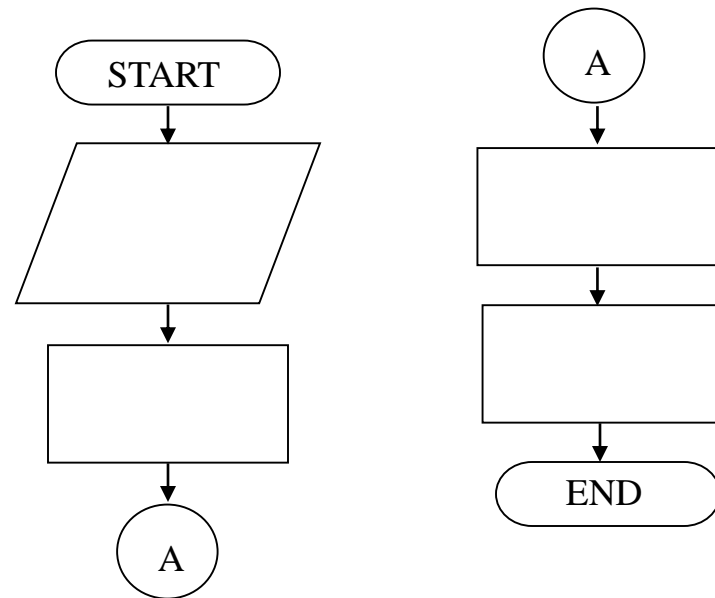
Connectors

- Sometimes a flowchart will not fit on one page.
- A connector (represented by a small circle) allows you to connect two flowchart segments.



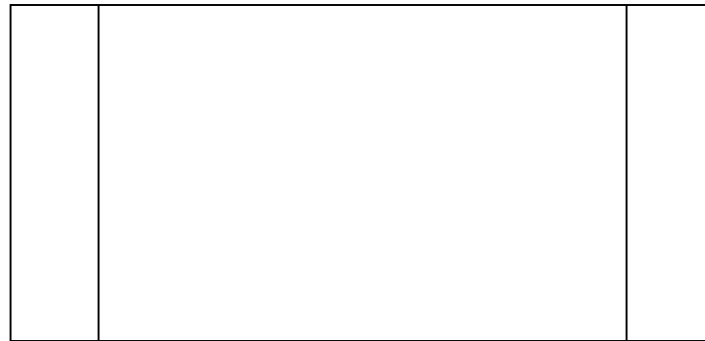
Connectors

- The “A” connector indicates that the second flowchart segment begins where the first segment ends.



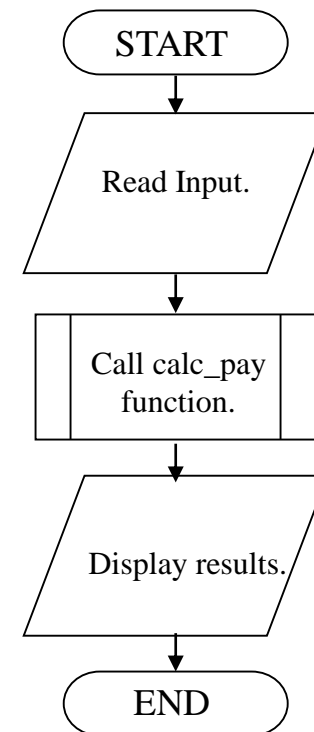
Modules

- A program module (such as a function in C++) is represented by a special symbol.



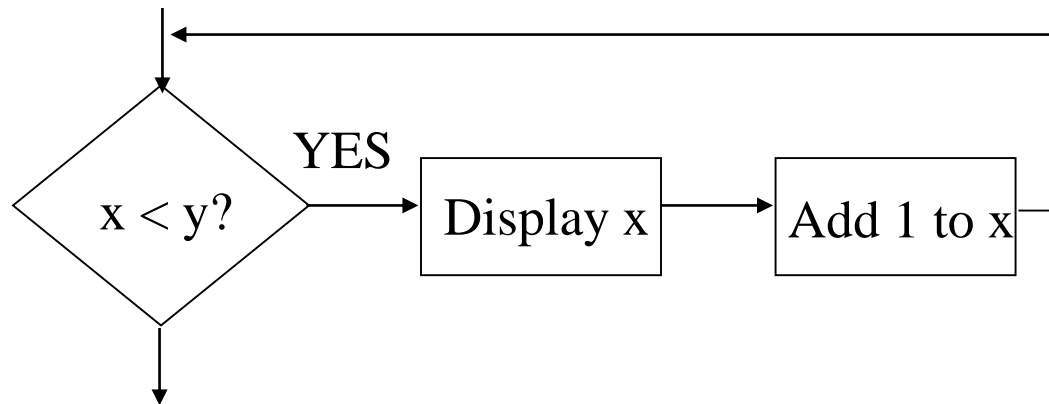
Modules

- The position of the module symbol indicates the point the module is executed.
- A separate flowchart can be constructed for the module.



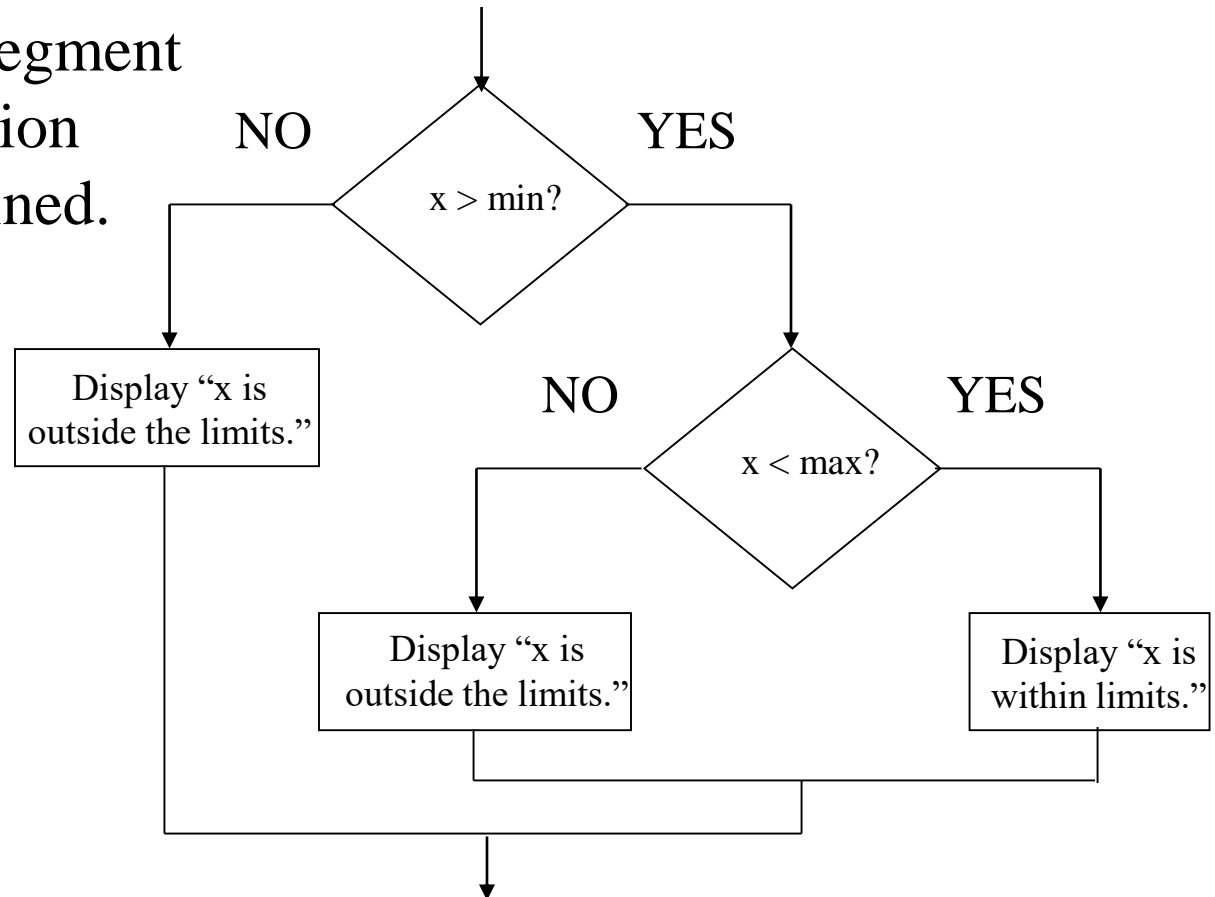
Combining Structures

- Structures are commonly combined to create more complex algorithms.
- The flowchart segment below combines a decision structure with a sequence structure.



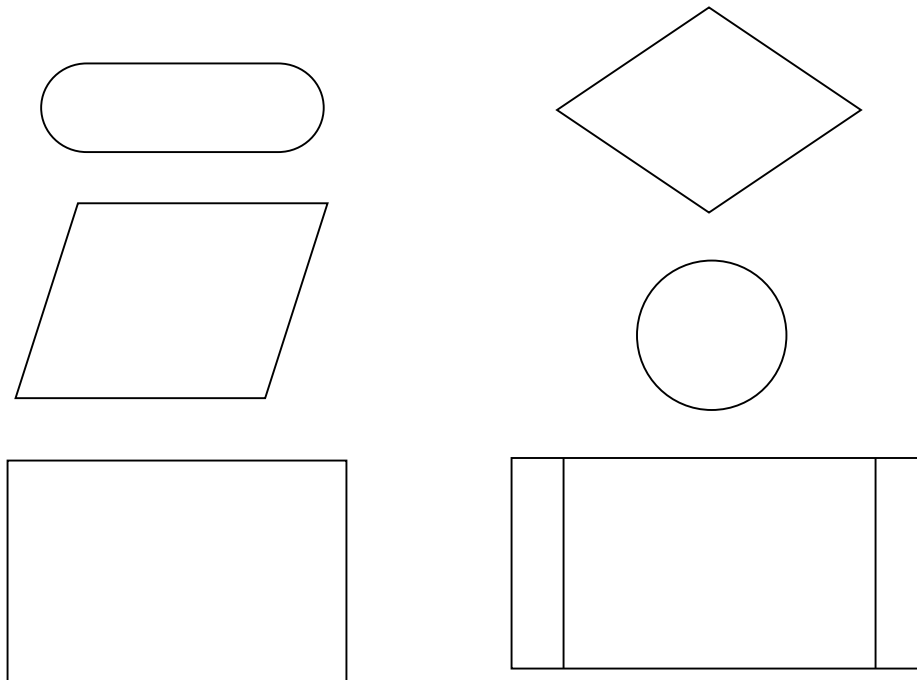
Combining Structures

- This flowchart segment shows two decision structures combined.



Review

- What do each of the following symbols represent?

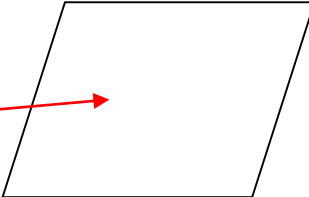



(Answer on next slide)

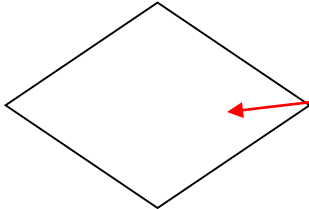
Answer

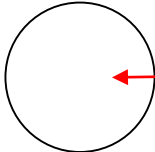
- What do each of the following symbols represent?

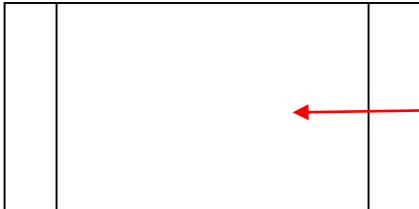
Terminal → 

Input/Output Operation → 

Process → 

Decision → 

Connector → 

Module → 

Review

- Name the four flowchart structures.

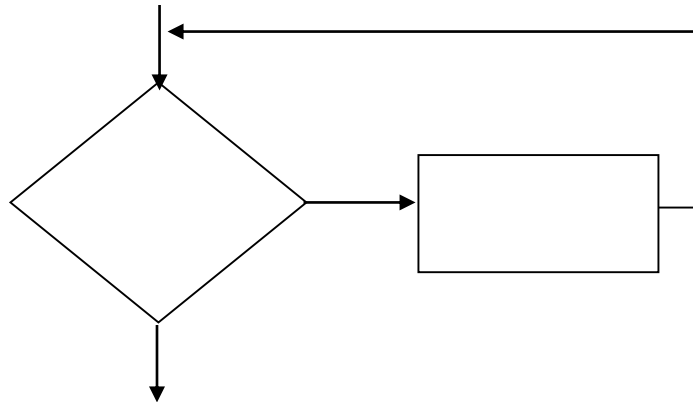
(Answer on next slide)

Answer

- Sequence
- Decision
- Repetition
- Case

Review

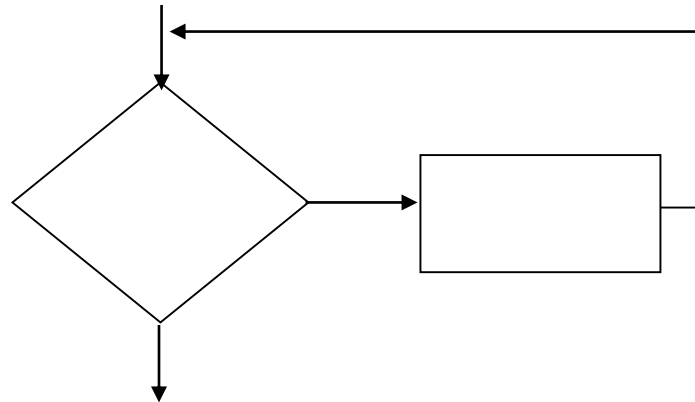
- What type of structure is this?



(Answer on next slide)

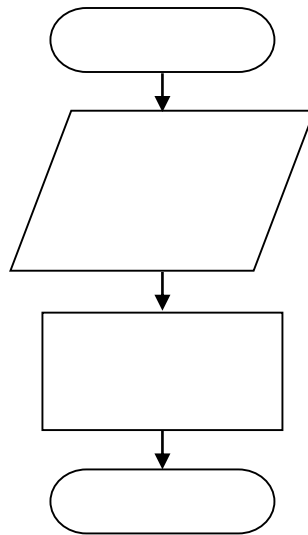
Answer

- Repetition



Review

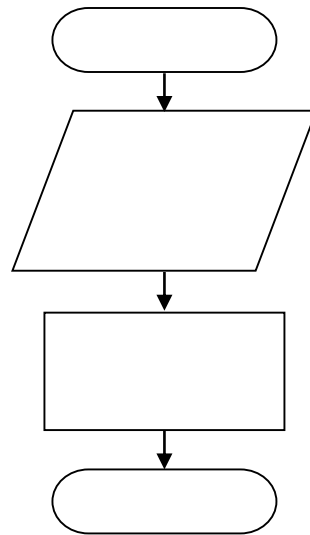
- What type of structure is this?



(Answer on next slide)

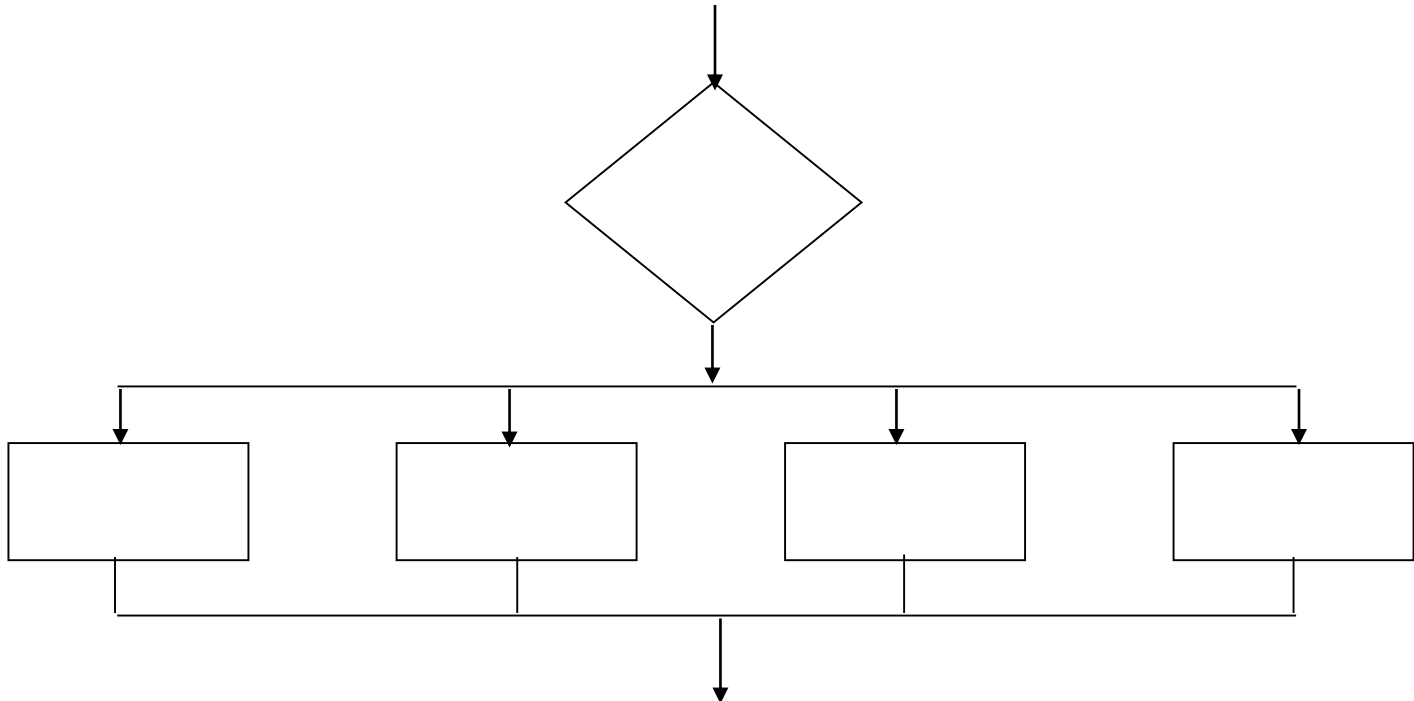
Answer

- Sequence



Review

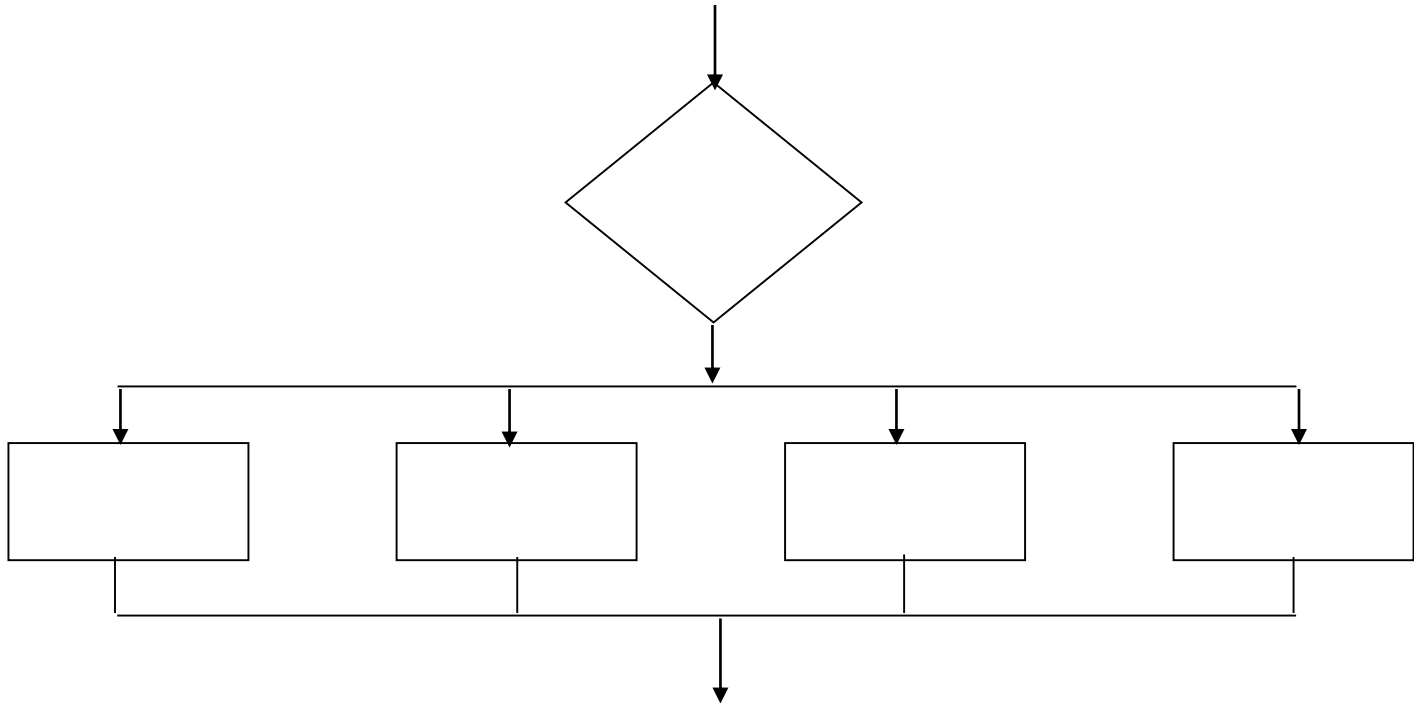
- What type of structure is this?



(Answer on next slide)

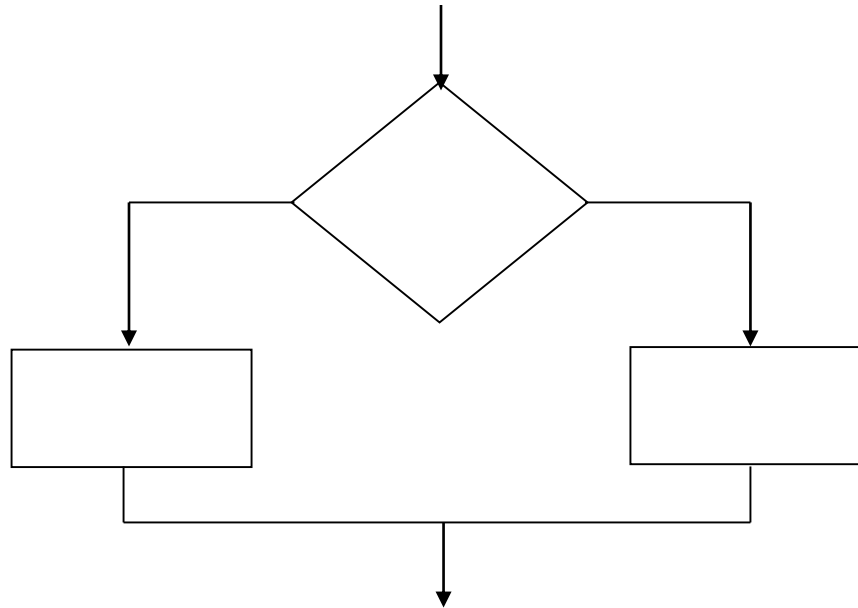
Answer

- Case



Review

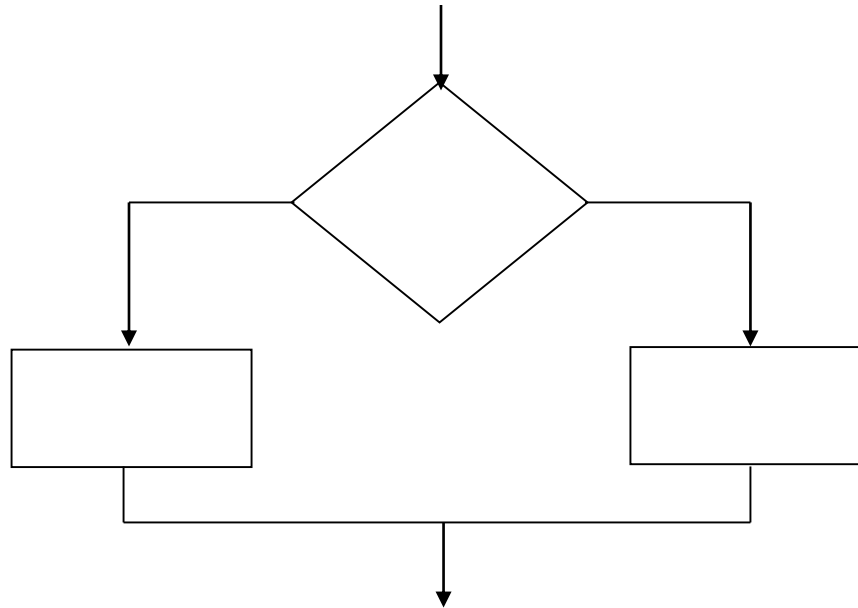
- What type of structure is this?



(Answer on next slide)

Answer

- Decision



Pseudocode & Algorithm

- **Example 1:** Write an algorithm to determine a student's final grade and indicate whether it is passing or failing. The final grade is calculated as the average of four marks.

Pseudocode & Algorithm

- Detailed Algorithm

Step 1: Input M1,M2,M3,M4

Step 2: $\text{GRADE} \leftarrow (M1+M2+M3+M4)/4$

Step 3: if (GRADE < 50) then

 Print “FAIL”

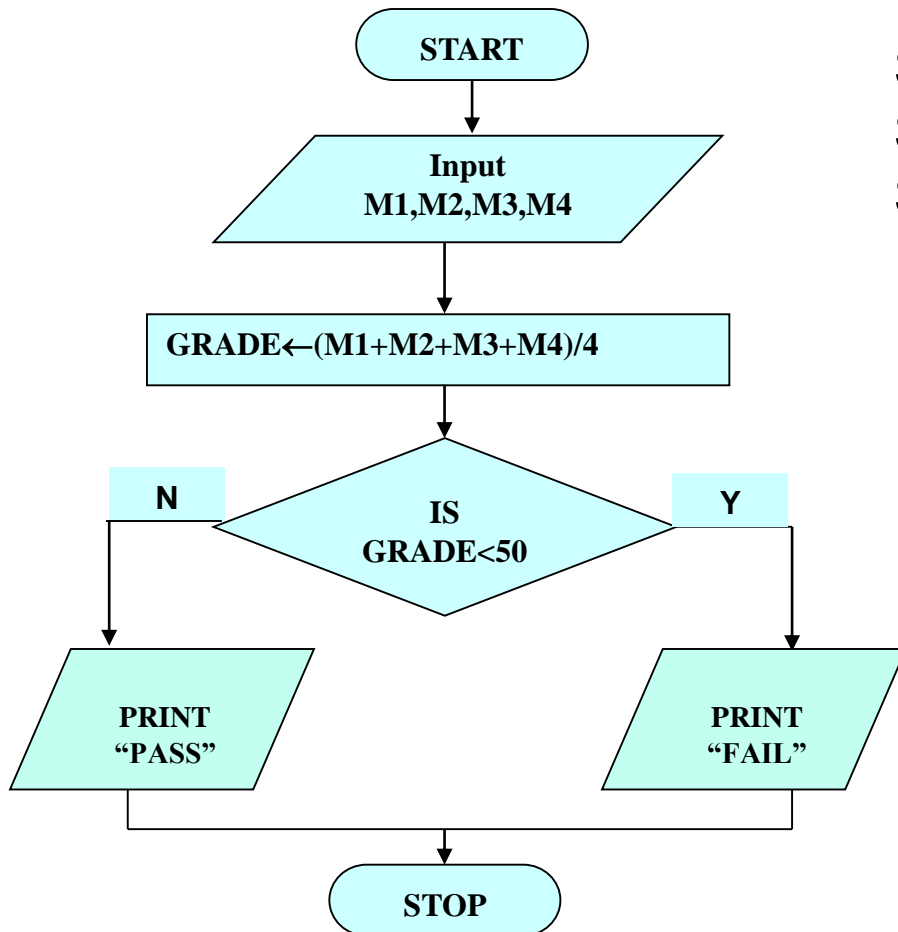
 else

 Print “PASS”

 endif

Example 1

Step 1: Input M1,M2,M3,M4
Step 2: $\text{GRADE} \leftarrow (M1+M2+M3+M4)/4$
Step 3: if (GRADE < 50) then
 Print "FAIL"
else
 Print "PASS"
endif



Example 2

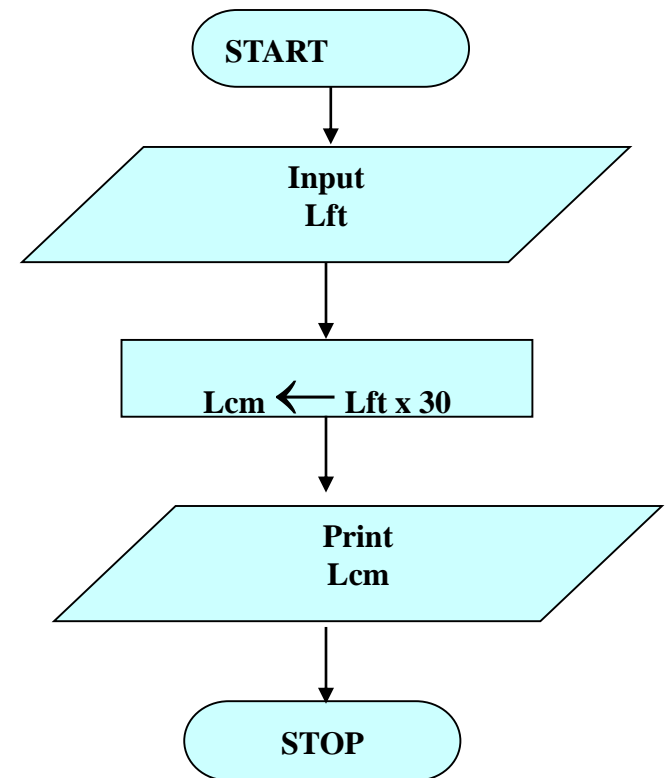
- Write an algorithm and draw a flowchart to convert the length in feet to centimeter.

Example 2

Algorithm

- Step 1: Input Lft
- Step 2: $Lcm \leftarrow Lft \times 30$
- Step 3: Print Lcm

Flowchart



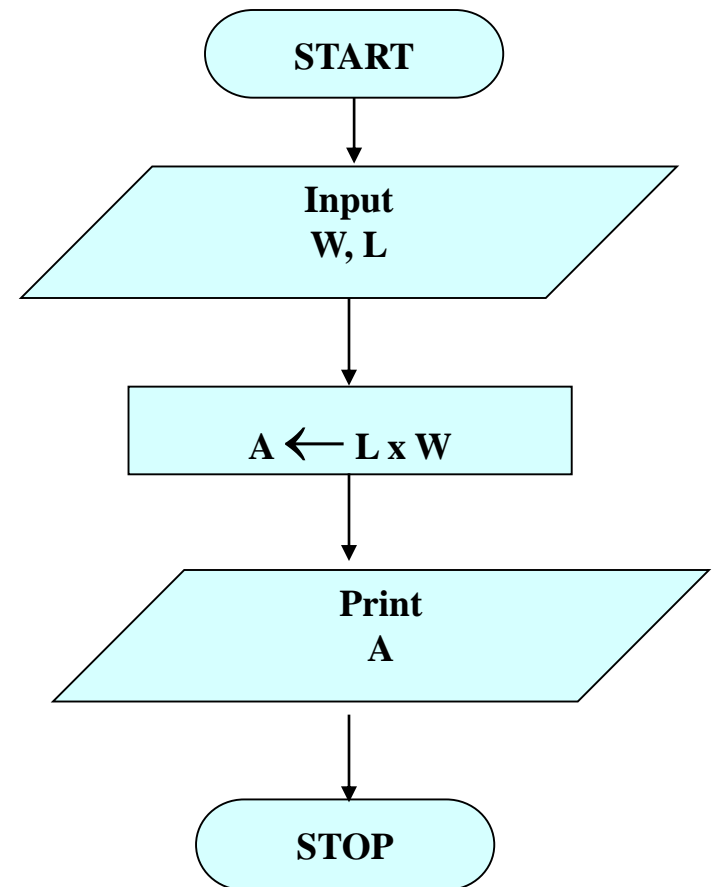
Example 3

Write an algorithm and draw a flowchart that will read the two sides of a rectangle and calculate its area.

Example 3

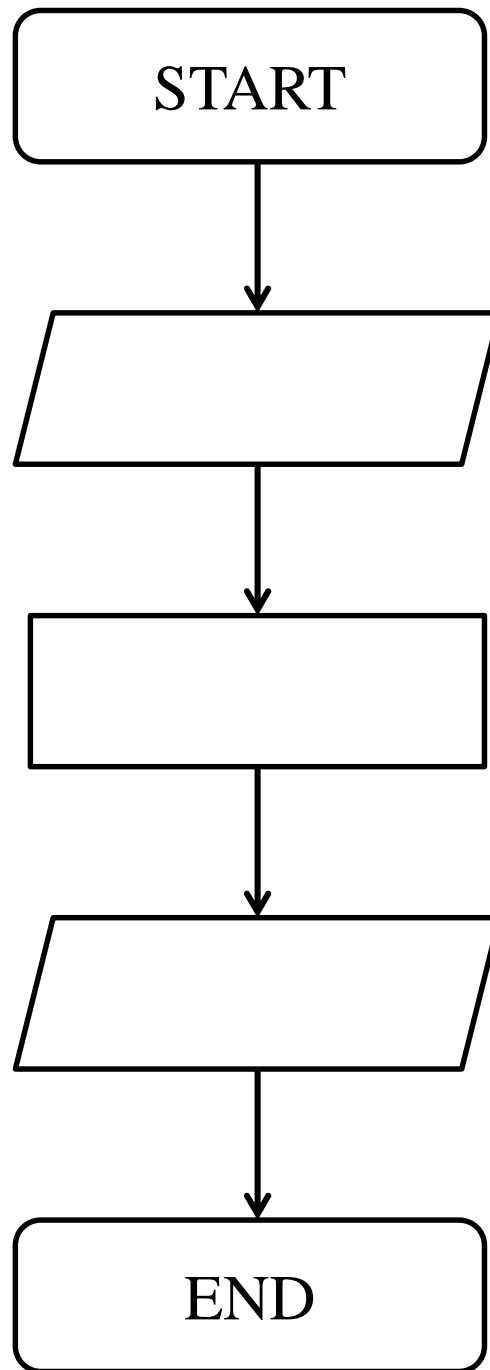
Algorithm

- Step 1: Input W,L
- Step 2: $A \leftarrow L \times W$
- Step 3: Print A



Example 4

- We want to create a flowchart that prints out double the number of the inputted value.
- On the following slide, a number of potential boxes you could use to correctly implement the algorithm.



- Pick the appropriate three of the following boxes that describe the algorithm as described.*

Read A

$B = A * 2$

Print A

Read B

$B = A / 2$

Print B

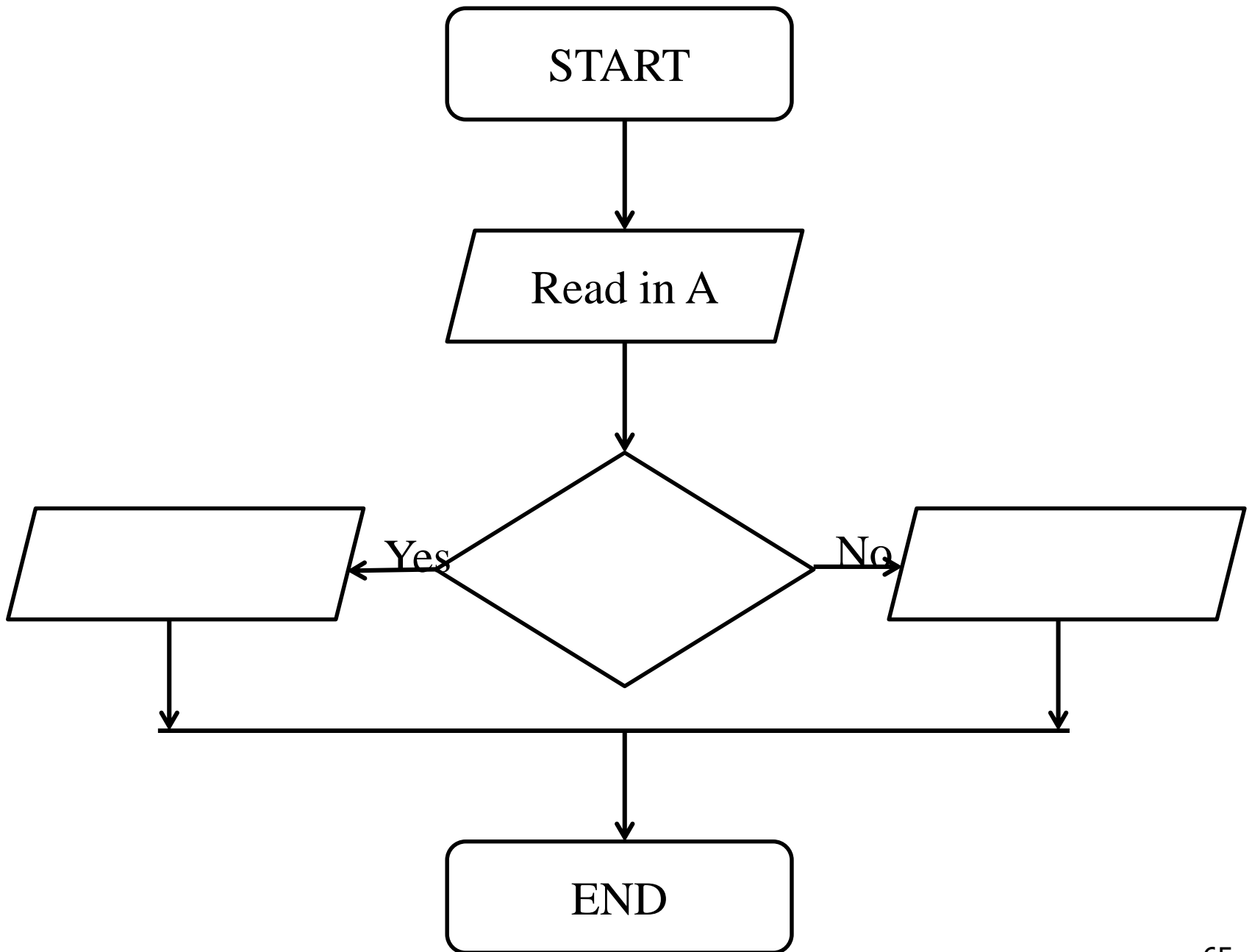
Read C

$B = A + 2$

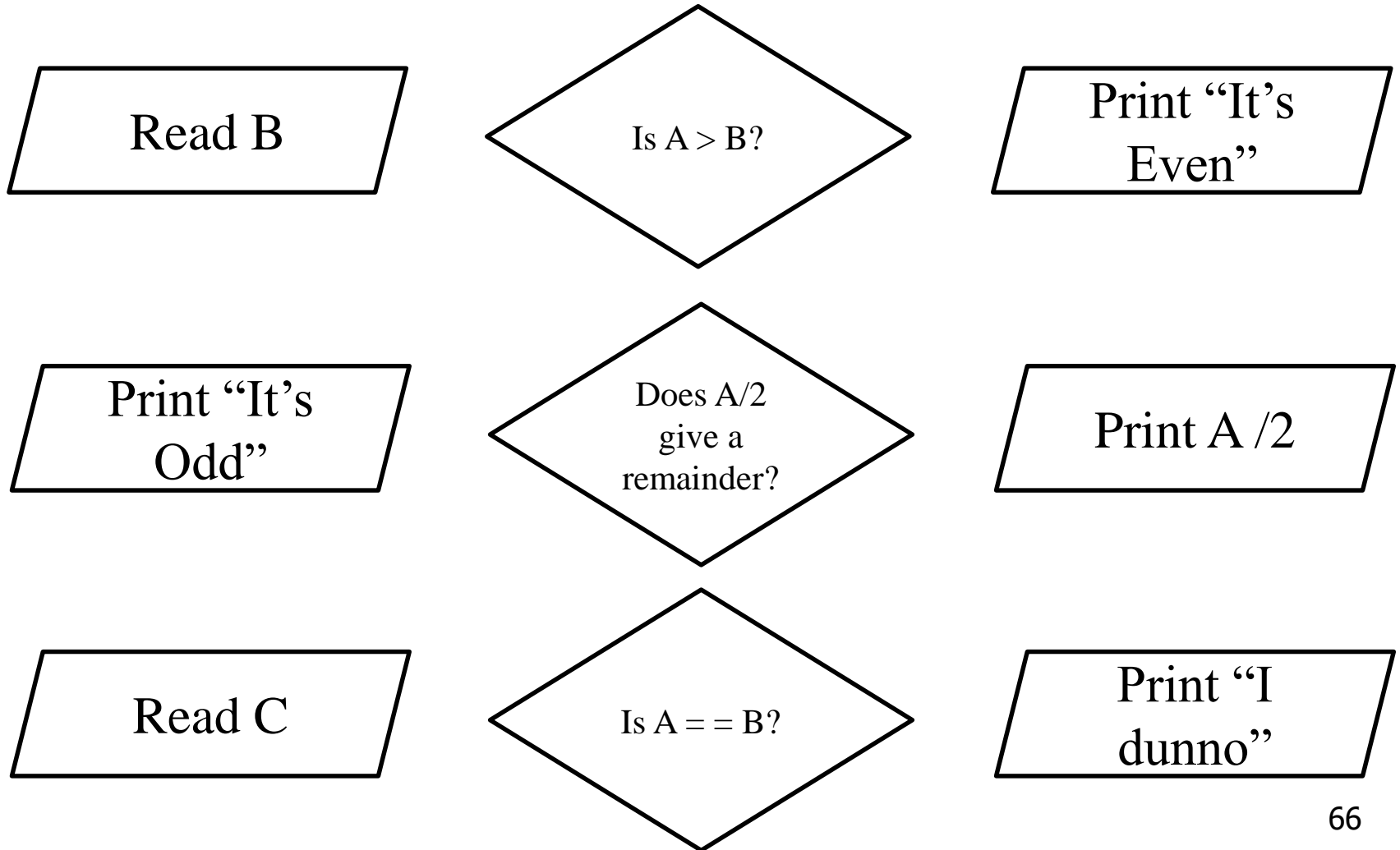
Print C

Example 5

- We want to create a flowchart that reads in a number, and checks if it is odd or even.
- On the following slide, a number of potential boxes you could use to correctly implement the algorithm.



- Pick the appropriate three of the following boxes that describe the algorithm as described.*



NESTED IFS

- One of the alternatives within an IF–THEN–ELSE statement
 - may involve further IF–THEN–ELSE statement

Example 6

- We want to create a flowchart that prints out the biggest of three inputted numbers

Example 6

Step 1: *Input* A, B, C

Step 2: *if* (A>B) *then*

if (A>C) *then*

 MAX \leftarrow A [A>B, A>C]

else

 MAX \leftarrow C [C>A>B]

endif

else

if (B>C) *then*

 MAX \leftarrow B [B>A, B>C]

else

 MAX \leftarrow C [C>B>A]

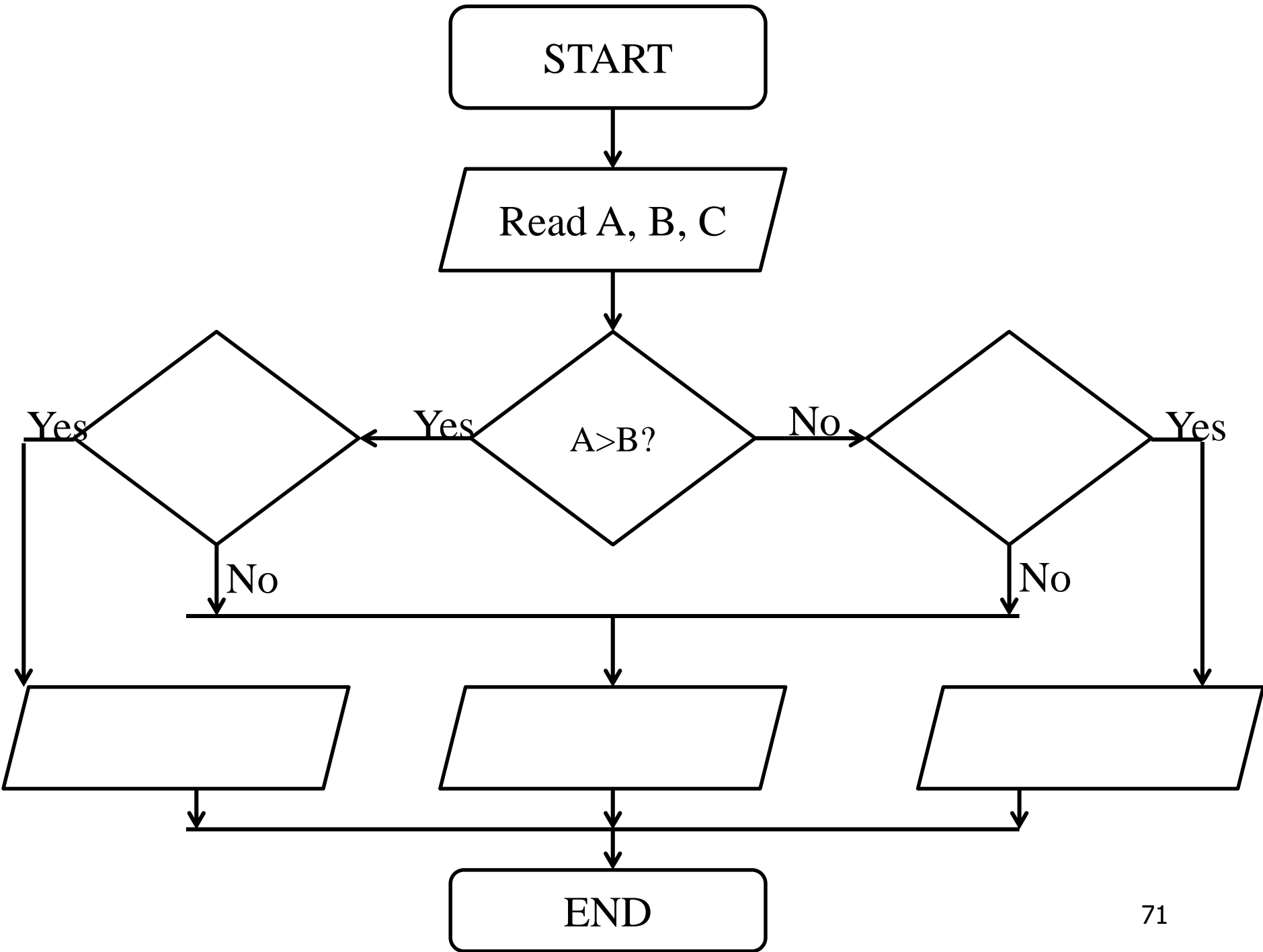
endif

endif

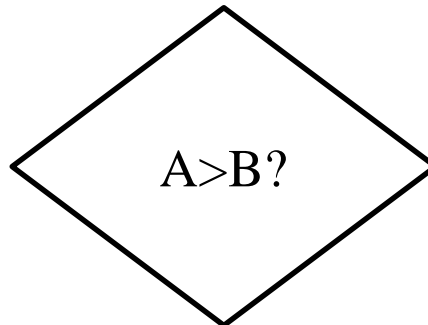
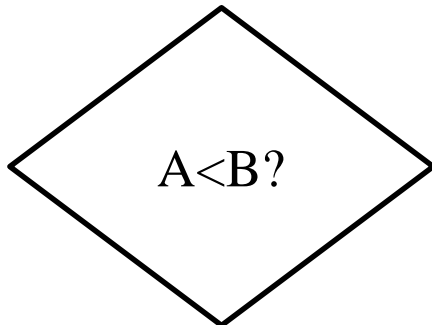
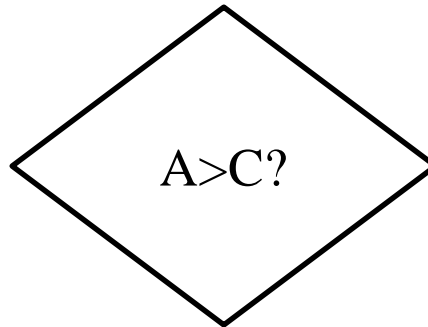
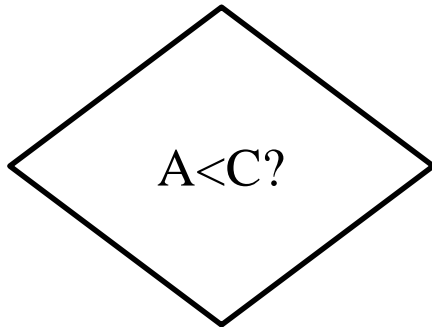
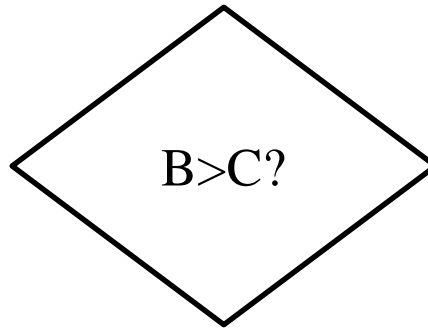
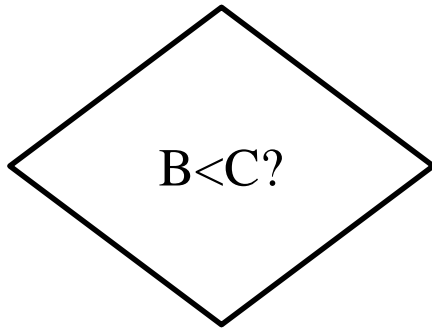
Step 3: *Print* “The largest number is”, MAX

Example 6

- **Flowchart: Draw the flowchart of the above Algorithm.**
- On the following slide, a number of potential boxes you could use to correctly implement the algorithm.

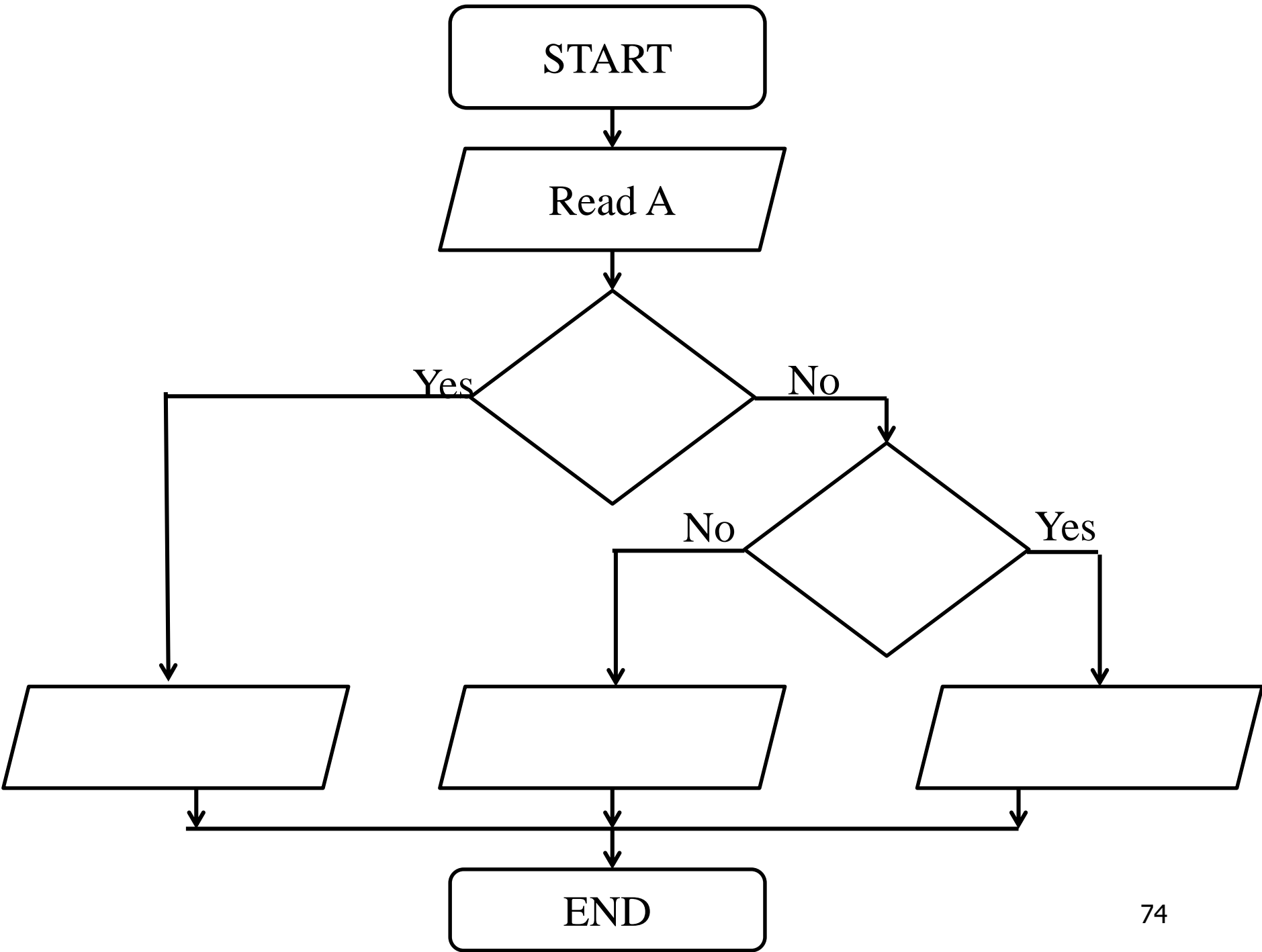


- Pick the appropriate three of the following boxes that describe the algorithm as described.*

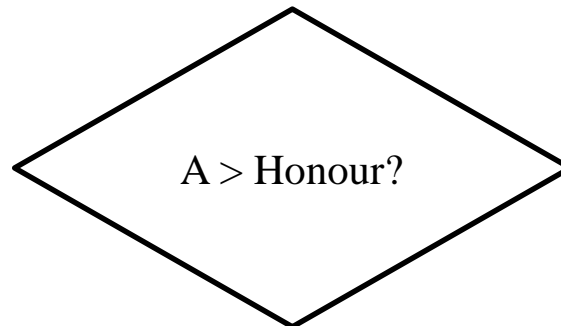
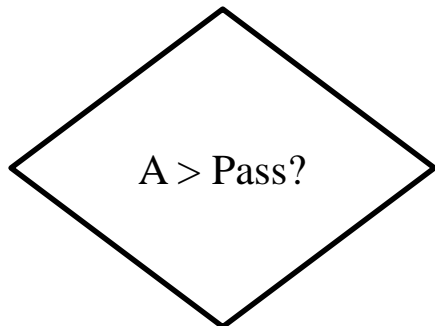
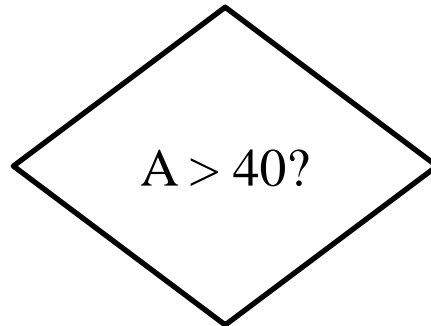
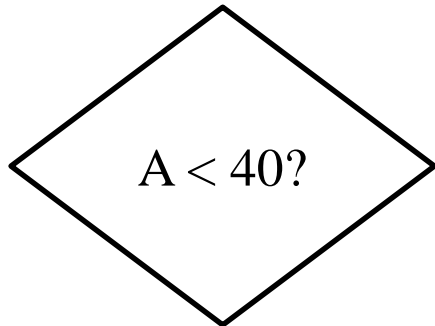
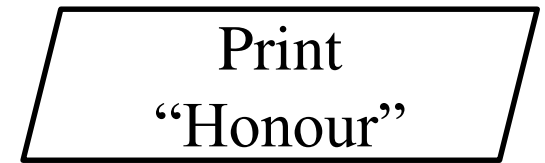
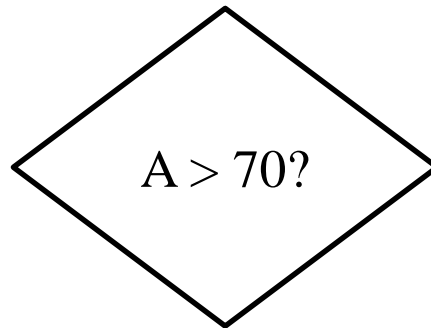
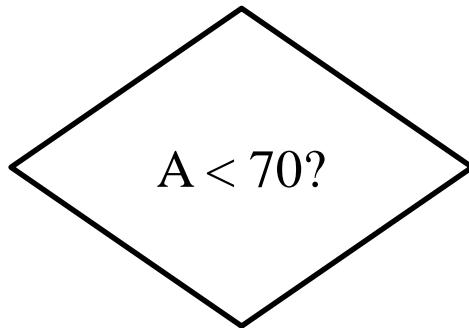


Example 7

- We want to create a flowchart that prints out the word “Honour” if the number input is greater than 70, if the number is less than 40 print out the word “Fail”, otherwise print out the word “Pass”.
- On the following slide, a number of potential boxes you could use to correctly implement the algorithm.

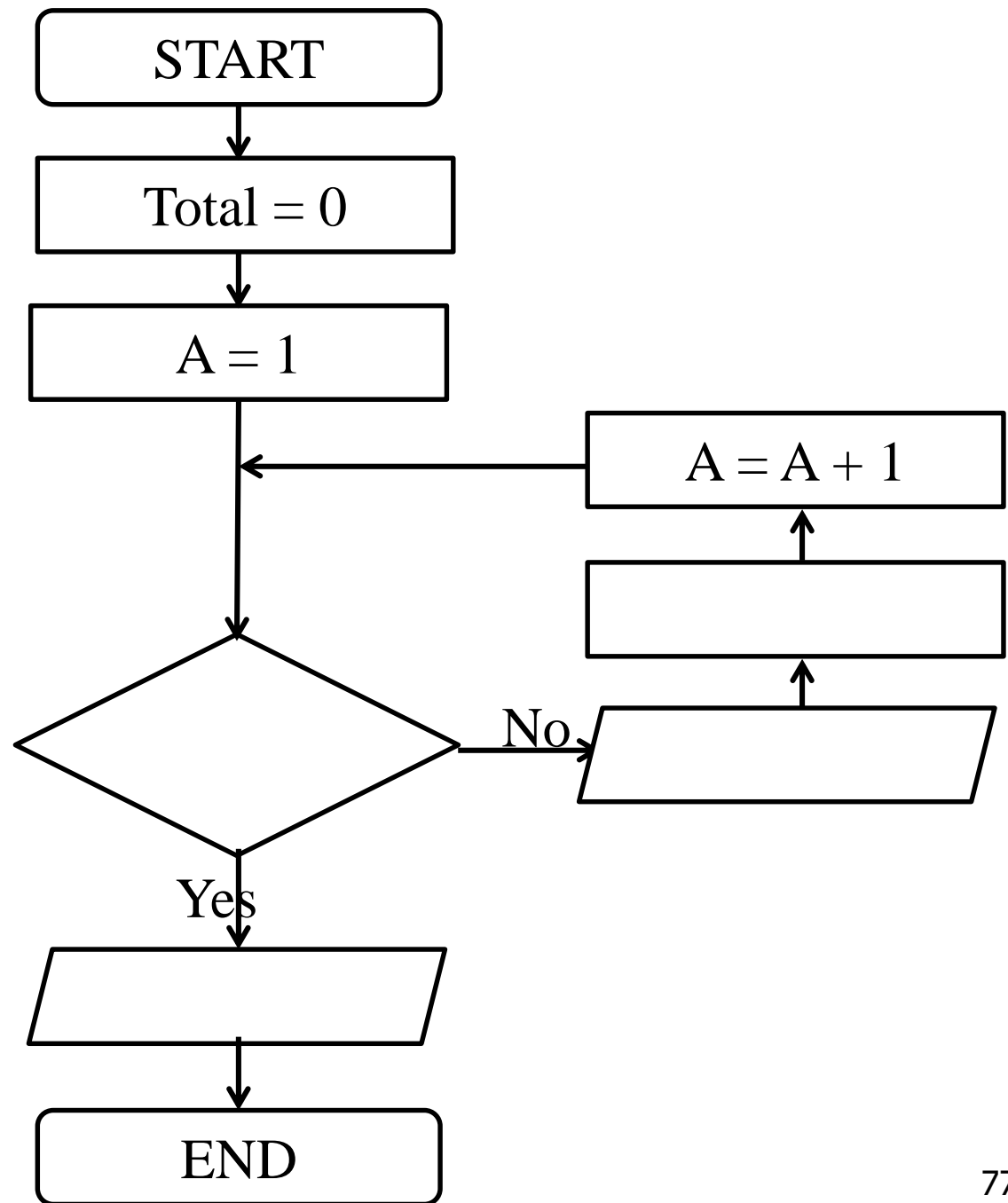


- Pick the appropriate three of the following boxes that describe the algorithm as described*

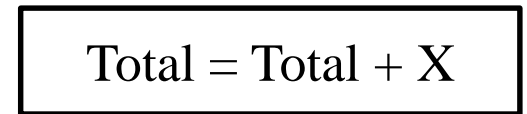
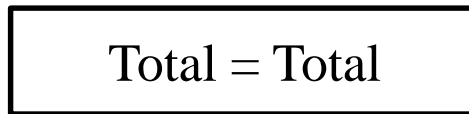
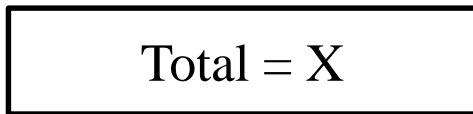
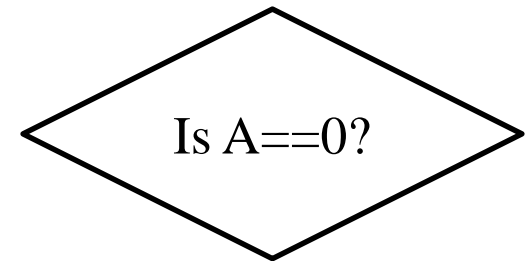
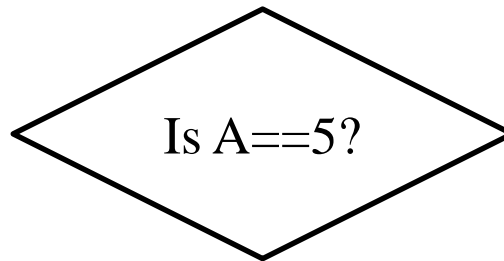
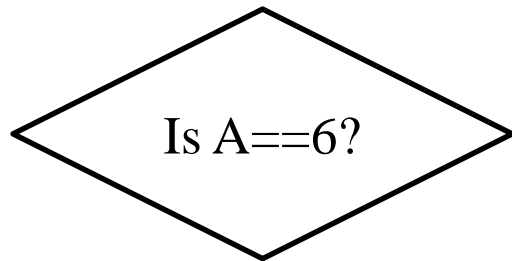


Example 8

- We want to create a flowchart that prints out the average value of five numbers input in.
- On the following slide, a number of potential boxes you could use to correctly implement the algorithm.



- Pick the appropriate three of the following boxes that describe the algorithm as described.*



Example 9

- Write an algorithm and draw a flowchart that will calculate the roots of a quadratic equation

$$ax^2 + bx + c = 0$$

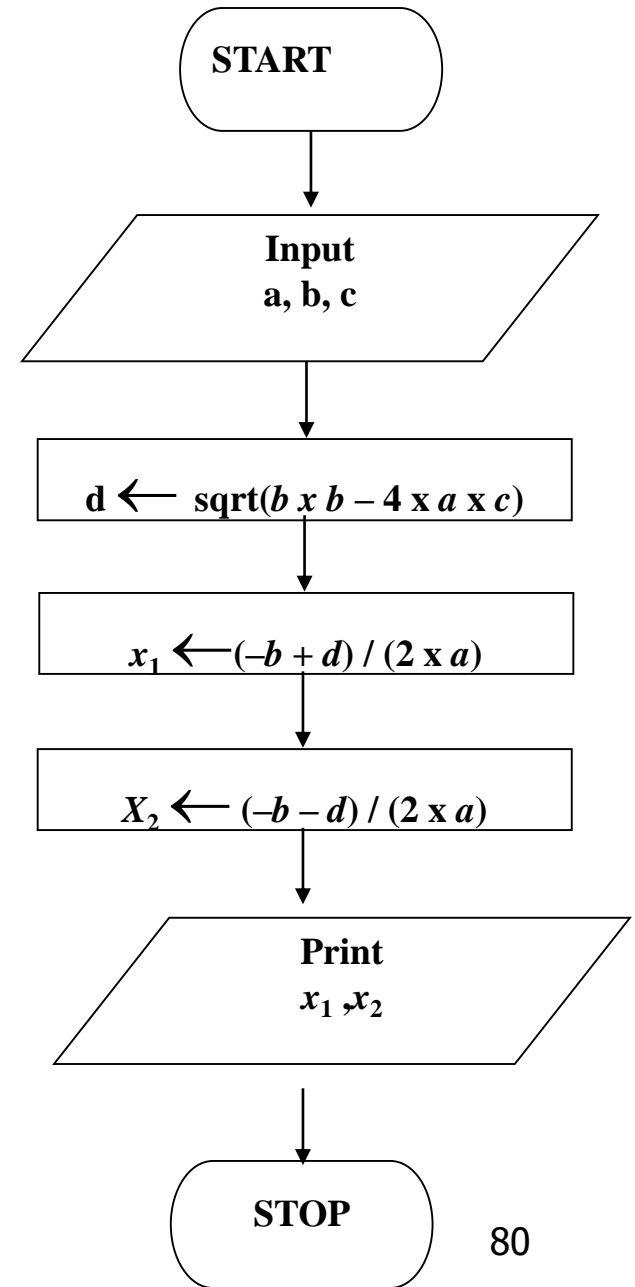
- Hint: $\mathbf{d} = \text{sqrt}(b^2 - 4ac)$,
- and the roots are:

$$\mathbf{x1} = (-b + d)/2a \quad \text{and} \quad \mathbf{x2} = (-b - d)/2a$$

Example 9

- **Algorithm:**

- Step 1: Input a, b, c
- Step 2: $d \leftarrow \text{sqrt}(b \times b - 4 \times a \times c)$
- Step 3: $x_1 \leftarrow (-b + d) / (2 \times a)$
- Step 4: $x_2 \leftarrow (-b - d) / (2 \times a)$
- Step 5: Print x_1, x_2



Example 10

- Write an algorithm that reads two values, determines the largest value and prints the largest value with an identifying message.

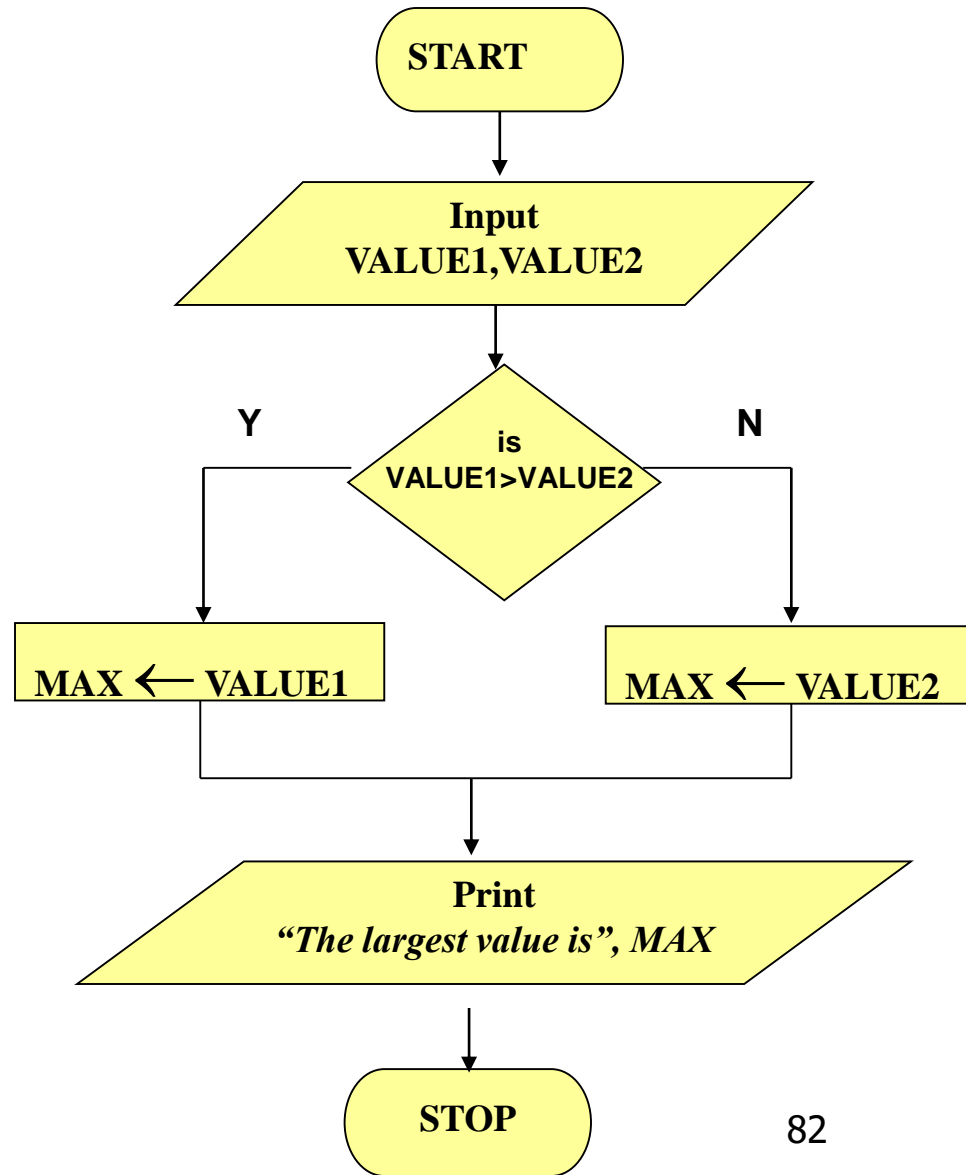
Example 10

ALGORITHM

Step 1: *Input* VALUE1, VALUE2

Step 2: *if* (VALUE1 > VALUE2)
 then
 MAX ← VALUE1
 else
 MAX ← VALUE2
 endif

Step 3: *Print* “The largest value is”, MAX



Example 11

Write an algorithm and draw a flowchart to

- a) read an employee name (NAME), overtime hours worked (OVERTIME), hours absent (ABSENT) and
- b) determine the bonus payment (PAYMENT).

Example 11

Bonus Schedule	
OVERTIME – $(2/3)*\text{ABSENT}$	Bonus Paid
>40 hours	\$50
>30 but \leq 40 hours	\$40
>20 but \leq 30 hours	\$30
>10 but \leq 20 hours	\$20
\leq 10 hours	\$10

Step 1: *Input* NAME,OVERTIME,ABSENT

Step 2: *if* (OVERTIME–(2/3)*ABSENT > 40) *then*

 PAYMENT ← 50

else if (OVERTIME–(2/3)*ABSENT > 30) *then*

 PAYMENT ← 40

else if (OVERTIME–(2/3)*ABSENT > 20) *then*

 PAYMENT ← 30

else if (OVERTIME–(2/3)*ABSENT > 10) *then*

 PAYMENT ← 20

else

 PAYMENT ← 10

endif

Step 3: *Print* “Bonus for”, NAME “is \$”, PAYMENT

Example 11

- **Flowchart: Draw the flowchart of the above algorithm?**

Example 12

Express an algorithm to get two numbers from the user (dividend and divisor), testing to make sure that the divisor number is not zero, and displaying their quotient using a *flowchart*.

Example 12 Answer

- **Step 1** – Declare variables – dividend, divisor, quotient
- **Step 2** – Prompt user to get dividend
- **Step 3** – Store values in dividend variable
- **Step 4** – Prompt user to get divisor
- **Step 5** – Store value in divisor variable
- **Step 6** – Display dividend and divisor
- **Step 7 - Loop**
 - Selection: If divisor is equal to zero
 - Display error message, “divisor must be non-zero” and
 - go back to step 4
 - **Step 8** - Calculate quotient as dividend/divisor
 - **Step 9** - Display quotient

