PROGRAMMING METHODOLOGY
(PHƯƠNG PHÁP LẬP TRÌNH)

UNIT 2: Algorithmic Problem Solving

Acknowledgement

- The contents of these slides have origin from School of Computing, National University of Singapore.
- We greatly appreciate support from Mr. Aaron Tan Tuck Choy for kindly sharing these materials.

Policies for students

- These contents are only used for students PERSONALLY.
- Students are NOT allowed to modify or deliver these contents to anywhere or anyone for any purpose.

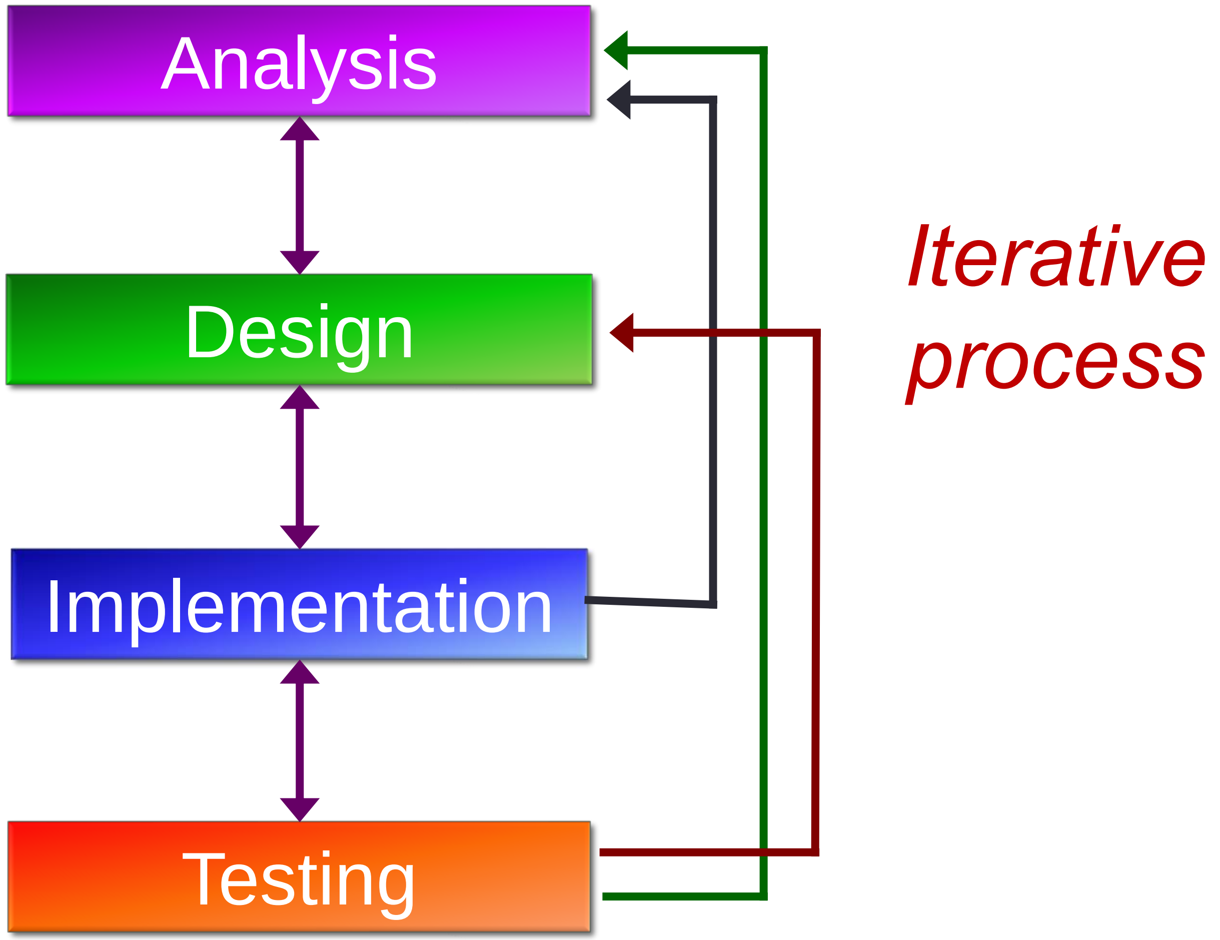
Recording of modifications

- Currently, there are no modification on these contents.

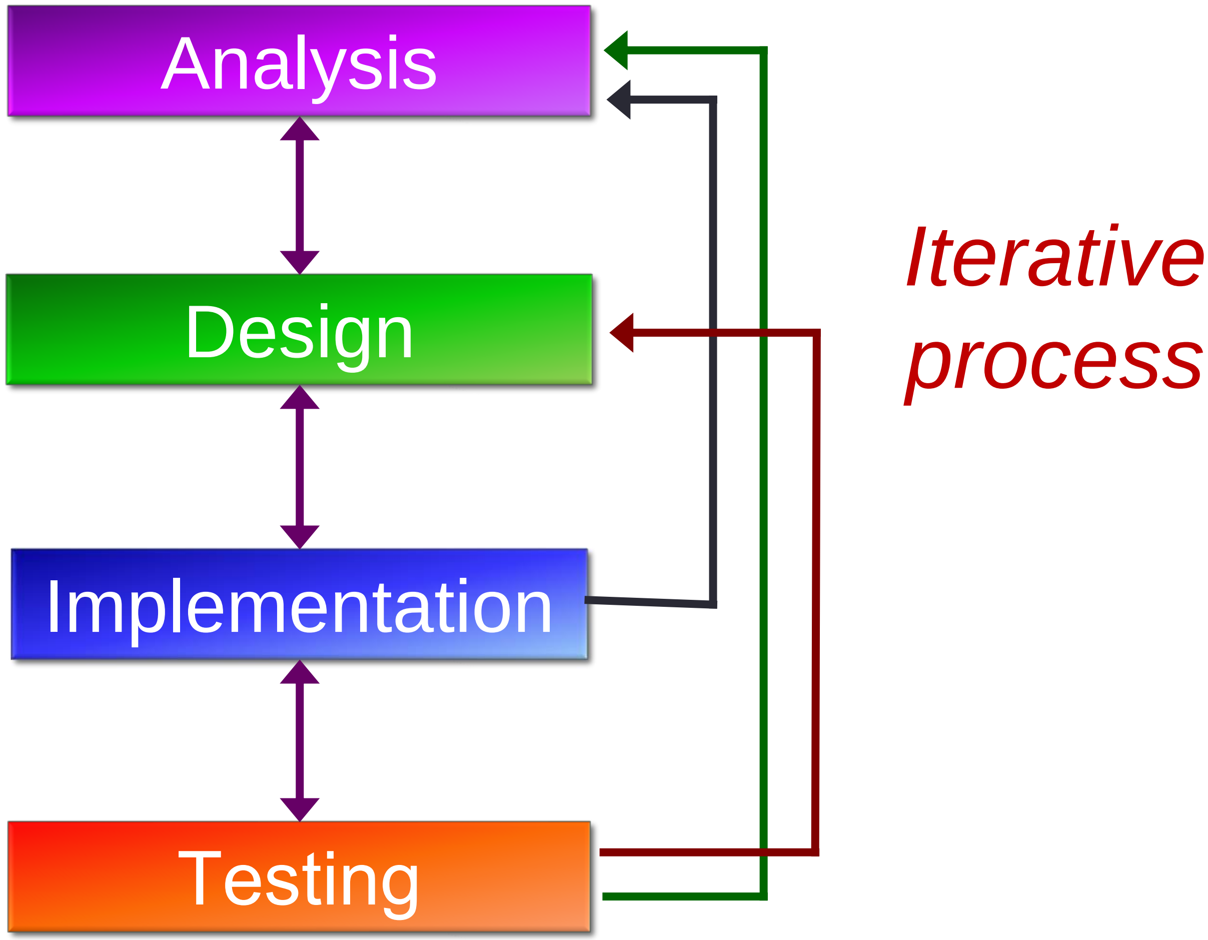
Unit 2: Algorithmic Problem Solving

1. Problem Solving Process
2. Algorithm
3. Control Structures
4. Examples of Pseudocodes
5. Euclid’s Algorithm

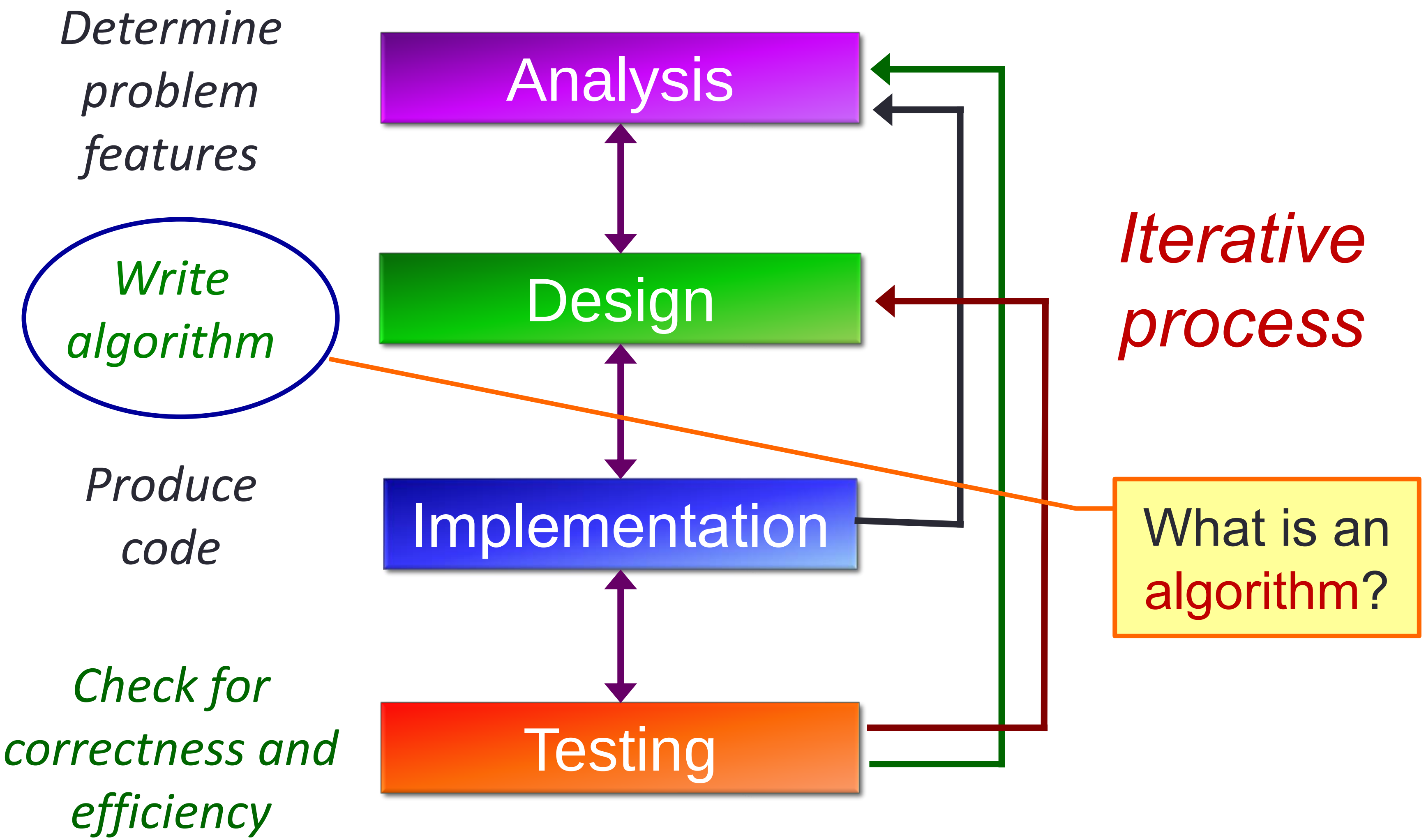
Problem Solving Process



Problem Solving Process



Algorithmic Problem Solving Process



Algorithm (1/3)

- An **algorithm** is a well-defined computational procedure consisting of *a set of instructions*, that takes some value or set of values as *input*, and produces some value or set of values as *output.*

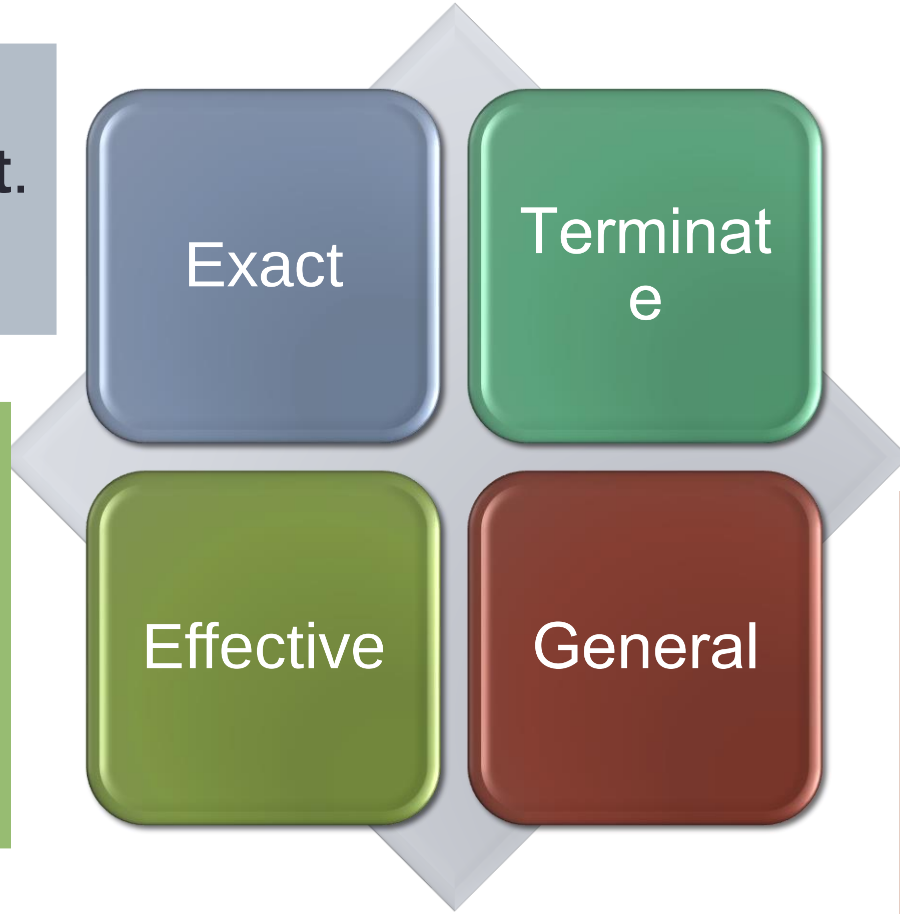
'Algorithm' stems from 'Algoritmi', the Latin form of al-Khwārizmī, a Persian mathematician, astronomer and geographer.
Source: http://en.wikipedia.org/wiki/Algorithm

Algorithm (2/3)

- An **algorithm** has these properties:

Each step must be **exact**.
(Or it will not be precise.)

The algorithm must **terminate**.
(Or no solution will be obtained.)



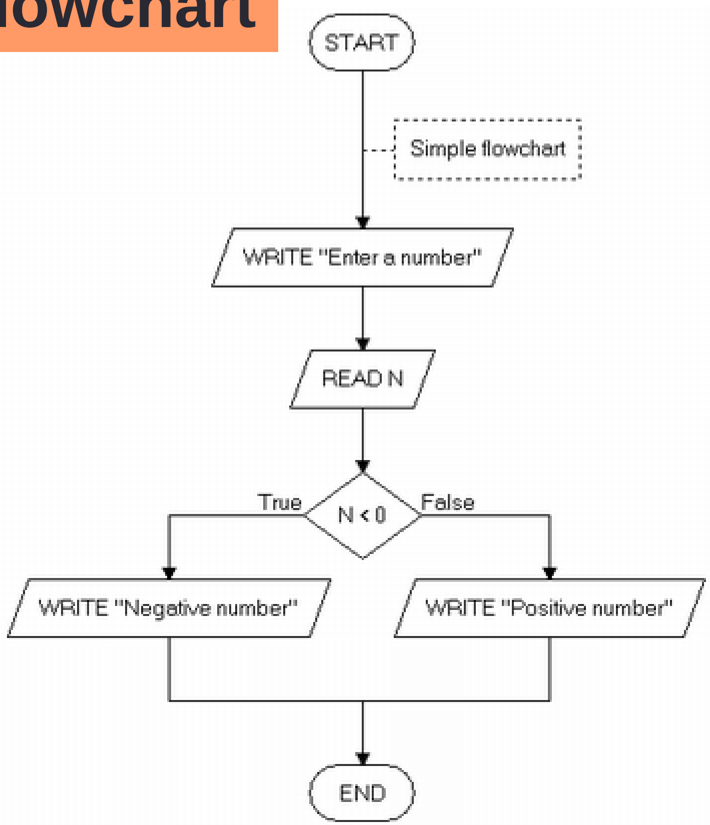
The algorithm must be **effective**.
(i.e. it must solve the problem.)

The algorithm must be **general**.
(Within the constraints of the system/language.)

Algorithm (3/3)

- Ways of representing an algorithm:

Flowchart



Pseudocode

PSEUDOCODE

```
set total to zero

get list of numbers

loop through each number in the list
    add each number to total
end loop

if number more than zero
    print "it's positive" message
else
    print "it's zero or less" message
end if
```

lynda.com

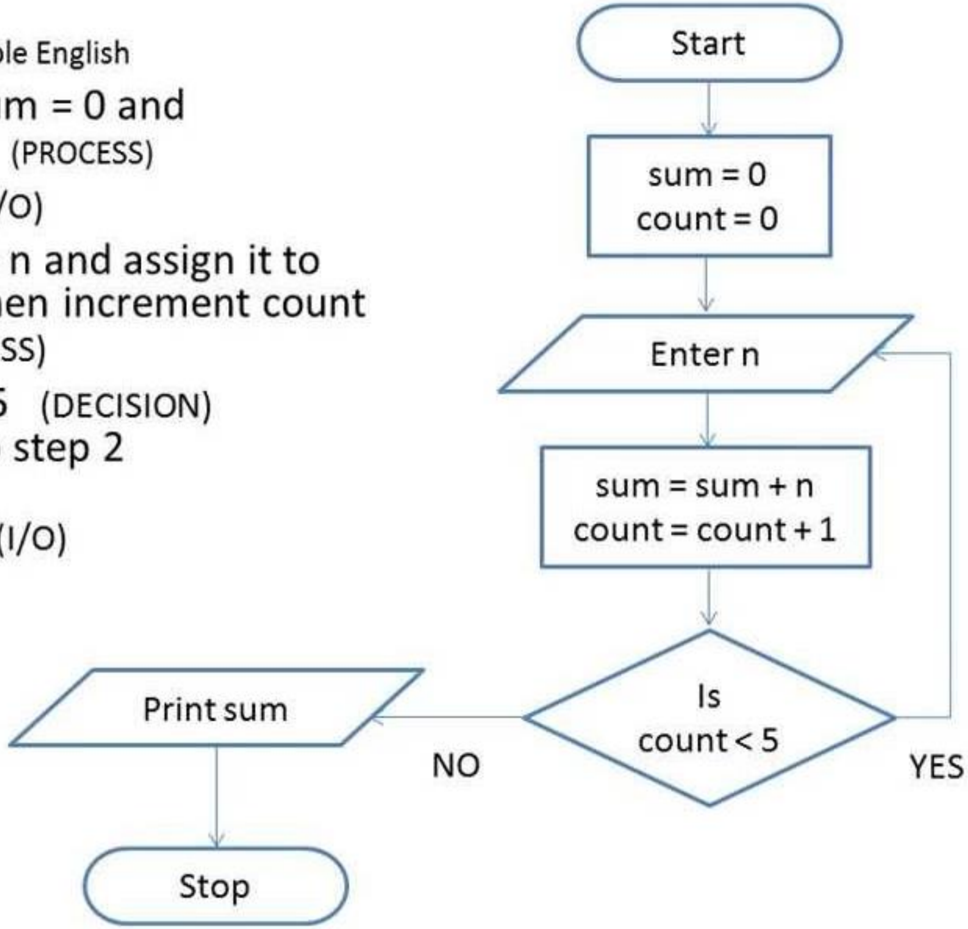
Algorithm: Example #1

Find the sum of 5 numbers

Algorithm in simple English

1. Initialize sum = 0 and count = 0 (PROCESS)
2. Enter n (I/O)
3. Find sum + n and assign it to sum and then increment count by 1 (PROCESS)
4. Is count < 5 (DECISION)
   if YES go to step 2
   else
   Print sum (I/O)

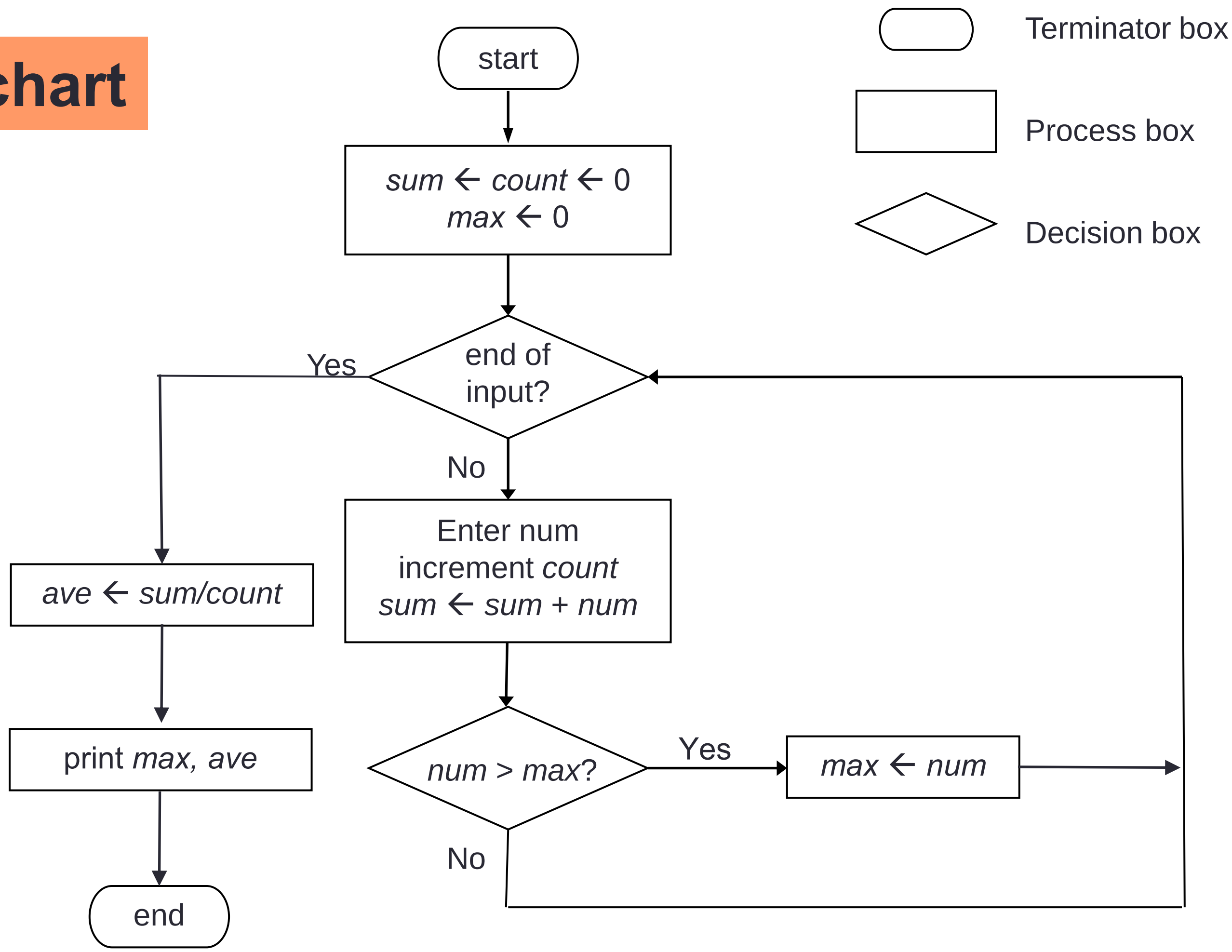
Flowchart



Algorithm: Example #2 (1/2)

- Find maximum and average of a list of numbers:

Flowchart



Algorithm: Example #2 (2/2)

- Find maximum and average of a list of numbers:

Pseudocode

The need to initialise variables.

```
sum ← count ← 0          // sum = sum of numbers
                         // count = how many numbers are entered?

max ← 0                  // max to hold the largest value eventually

for each num entered,
    count ← count + 1
    sum ← sum + num
    if num > max
        then max ← num

ave ← sum / count

print max, ave
```

The need to indent.

Are there any errors in this algorithm?

Algorithm: Pseudocode

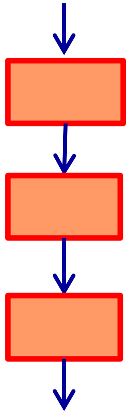
- We will write algorithms in pseudocode instead of flowchart as the former is more succinct
- However, there are no standard rules on how pseudocodes should look like
- General guidelines:
  - Every step must be unambiguous, so that anybody is able to hand trace the pseudocode and follow the logic flow
  - Use a combination of English (keep it succinct) and commonly understood notations (such as $\leftarrow$ for assignment in our previous example)

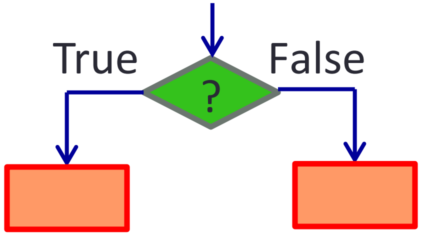
Control Structures (1/2)

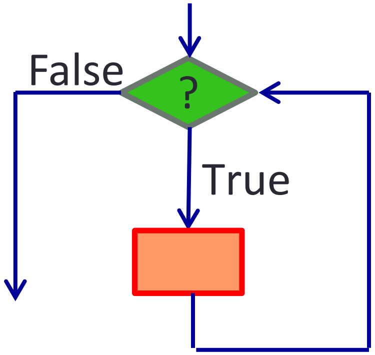
- An algorithm is a set of instructions, which are followed sequentially by default.
- However, sometimes we need to change the default sequential flow.
- We study 3 control structures.

Control Structures (2/2)

Sequence	• Default
Selection	• Also called branching
Repetition	• Also called loop



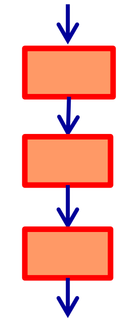




Data Representation

- Internal representation: bits (binary digits) 0 and 1
- 1 byte = 8 bits
- In programming, we need variables to hold data. A variable has an associated data type and occupies memory space. In the following slides, variables are shown as boxes.
- Some data types in C (list is not exhaustive)
  - Integers: int, short, long (int is most common)
  - Real numbers: float, double
  - Characters: char
- Self-reading: Lesson 1.4 in reference book

Control Structures: Sequence (1/2)

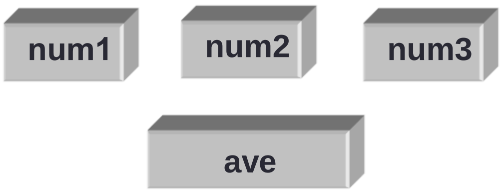


- Task: Compute the average of three integers

A possible algorithm:

> enter values for *num1*, *num2*, *num3*
> *ave* ← (*num1* + *num2* + *num3*) / 3
> print *ave*

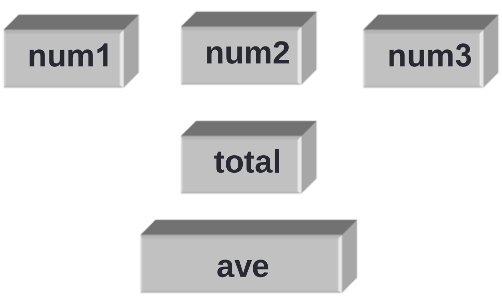
Variables used:



Another possible algorithm:

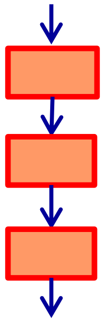
> enter values for *num1*, *num2*, *num3*
> *total* ← (*num1* + *num2* + *num3*)
> *ave* ← *total* / 3
> print *ave*

Variables used:



Each box represents a variable.
Important concepts: Each variable has a unique name and contains a value.

Control Structures: Sequence (2/2)

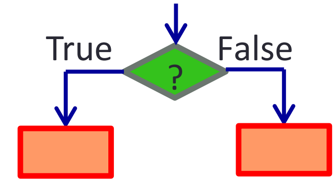


- Task: Compute the average of three integers
- How the program might look like

Unit2_prog1.c

```c
// This program computes the average of 3 integers
#include <stdio.h>

int main(void) {
   int num1, num2, num3;
   float ave;

   printf("Enter 3 integers: ");
   scanf("%d %d %d", &num1, &num2, &num3);

   ave = (num1 + num2 + num3) / 3.0;
   printf("Average = %.2f\n", ave);

   return 0;
}
```

Control Structures: Selection (1/3)



- Task: Arrange two integers in ascending order (sort)

Algorithm A:

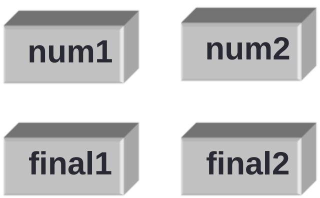
enter values for *num1*, *num2*

// Assign smaller number into *final1*,
// and larger number *into final2*
if (*num1* < *num2*)
then *final1* ← *num1*
final2 ← *num2*
else *final1* ← *num2*
final2 ← *num1*

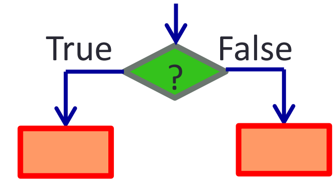
// Transfer values in *final1*, *final2* back to *num1*, *num2*
num1 ← *final1*
num2 ← *final2*

// Display sorted integers
print *num1*, *num2*

Variables used:



Control Structures: Selection (2/3)

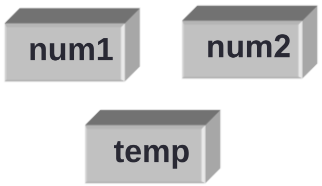


- Task: Arrange two integers in ascending order (sort)

Algorithm B:

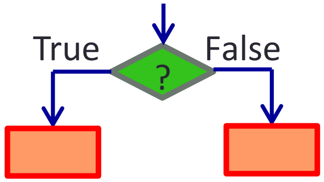
```
enter values for num1, num2

// Swap the values in the variables if necessary
if (num2 < num1)
   then  temp ← num1
         num1 ← num2
         num2 ← temp

// Display sorted integers
print num1, num2
```

Variables used:



Compare Algorithm A with Algorithm B.

Control Structures: Selection (3/3)

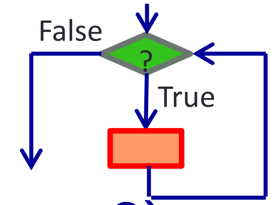


- How the program might look like for Algorithm B

Unit2_prog2.c

```
// This program arranges 2 integers in ascending order
#include <stdio.h>

int main(void) {
   int num1, num2, temp;

   printf("Enter 2 integers: ");
   scanf("%d %d", &num1, &num2);

   if (num2 < num1) {
      temp = num1; num1 = num2; num2 = temp;
   }
   printf("Sorted: num1 = %d, num2 = %d\n", num1, num2);

   return 0;
}
```

Control Structures: Repetition (1/3)



- Task: Find sum of positive integers up to *n* (assume *n*>0)

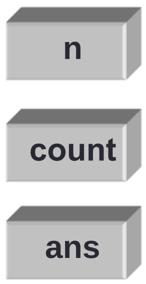
Algorithm:

enter value for *n*

// Initialise a counter *count* to 1, and *ans* to 0
count ← 1
ans ← 0
while (*count* <= *n*) do
    ans ← *ans* + *count*   **// add *count* to *ans***
    count ← *count* + 1   **// increase *count* by 1**

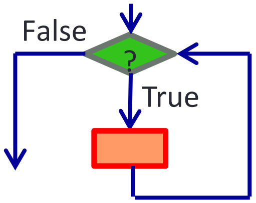
// Display answer
print *ans*

Variables used:



Initialisation is very important!

Control Structures: Repetition (2/3)



- Important to trace pseudocode to check its correctness

Algorithm:

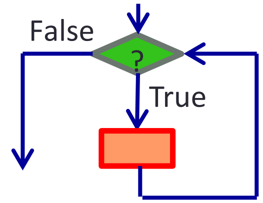
enter value for *n*
count ← 1
ans ← 0
while (*count* <= *n*) do
  ans ← *ans* + *count*
  count ← *count* + 1

// Display answer
print *ans*

Assume user enters **3** for *n*.

(*count* <= *n*)?	*count*	*ans*
	1	0
true	2	1
true	3	3
true	4	6
false		

Output: **6**



Control Structures: Repetition (3/3)

- How the program might look like

Unit2_prog3.c

```
// Computes sum of positive integers up to n
#include <stdio.h>

int main(void) {
   int n; // upper limit
   int count=1, ans=0; // initialisation

   printf("Enter n: ");
   scanf("%d", &n);

   while (count <= n) {
      ans += count;
      count++;
   }
   printf("Sum = %d\n", ans);

   return 0;
}
```

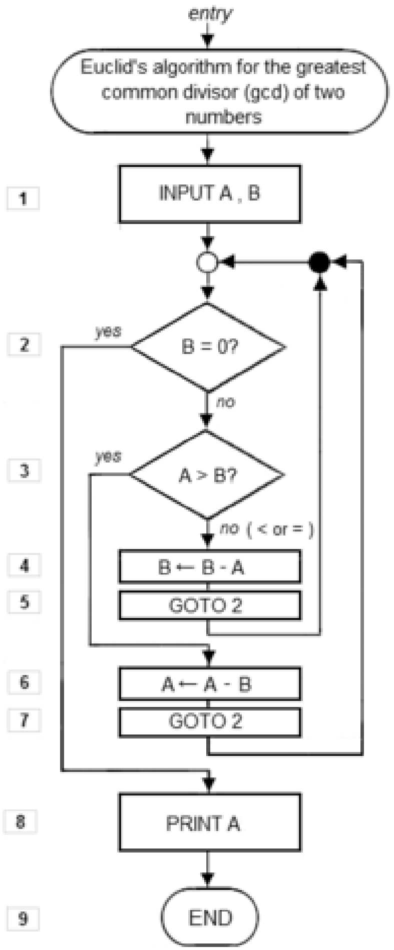
Euclid's Algorithm (1/3)

- To compute the greatest common divisor (GCD) of two integers
  - First documented algorithm by Greek mathematician Euclid in 300 B.C.
  - Also known as Euclidean Algorithm

1. Let A and B be integers with $A > B \geq 0$.
2. If $B = 0$, then the GCD is A and algorithm ends.
3. Otherwise, find q and r such that

   $$A = q.B + r \qquad \text{where } 0 \leq r < B$$

4. Replace A by B, and B by r. Go to step 2.



Euclid's Algorithm (2/3)

1. Let A and B be integers with $A > B \geq 0$.
2. If $B = 0$, then the GCD is A and algorithm ends.
3. Otherwise, find q and r such that

   $$A = q.B + r \qquad \text{where } 0 \leq r < B$$

4. Replace A by B, and B by r. Go to step 2.

- *q* is not important;
  r is the one that matters.

- *r* could be obtained by *A* modulo *B* (i.e. remainder of A / B)

- Assumption on $A > B$ unnecessary

- We will rewrite the algorithm

Euclid's Algorithm (3/3)

- Euclid's algorithm rewritten in modern form

```
// Assume A and B are non-negative
// integers, but not both zeroes.

Algorithm GCD(A, B) {
    while (B > 0) {
        r ← A modulo B
        A ← B
        B ← r
    }
    result is A
}
```

Let's trace GCD(12, 42)

(*B* > 0)?	*r*	*A*	*B*
		12	42
true	12	42	12
true	6	12	6
true	0	6	0
false			

Result: **6**

Summary

- In this unit, you have learned about
  - The process of algorithmic problem solving
  - The properties of an algorithm
  - The three control structures
  - How to write algorithms in pseudocode
  - Tracing algorithms to verify their correctness

End of File