

# ITC213: STRUCTURED PROGRAMMING

**Bhaskar Shrestha**  
National College of Computer Studies  
Tribhuvan University

# Lecture 02: Algorithms

Readings: Not Covered in Textbook

# Problem Solving Process

- System Design:
  - Find a set of sub-problems such that
  - each sub-problem is easy to solve; and
  - the desired problem can be solved by solving them
- Algorithm Design:
  - Find solution to each sub-problem
- Programming:
  - Translate algorithms into codes

# Algorithms

- Computing problems
  - Solved by executing a series of actions in a specific order
- Algorithm is a procedure determining
  - Actions to be executed
  - Order to be executed
  - Example: recipe
- Program control
  - Specifies the order in which statements are executed

# Algorithm: Definition

- An **algorithm** is a precise specification of a *sequence of instructions* to be carried out in order to solve a given problem
  - Each instruction tells what task is to be done
  - Specification of a sequence of instructions to do a job is used in many fields
    - A cooking recipe, The rules of how to play a game, VCR instructions, Description of a martial arts technique, Directions for driving from A to B, A knitting pattern, A car repair manual

# Recipe: Chocolate Chip Cookies (1/2)

- ***Ingredients:***

- 2 ¼ cups flour, 1 tsp salt, 1 tsp baking soda, 2 eggs, ¾ cup brown sugar, 1 tsp vanilla extract, ¾ cup granulated sugar, 1 cup soft butter, 12 oz. semi-sweet chocolate chips

- ***Steps:***

1. Preheat oven to 375 degrees
2. Combine flour, salt, baking soda, in bowl. Set mixture aside.
3. Combine sugars, butter, vanilla and beat until creamy.
4. Add eggs and beat.
5. Add dry mixture and mix well.
6. Stir in chocolate chips.
7. Drop mixture by teaspoons onto ungreased cookie sheet.
8. Bake 8 to 10 minutes

# Recipe: Chocolate Chip Cookies (2/2)

- Although the recipe given in the previous slide does not technically qualify as an algorithm but is similar to algorithm
  - It is a *sequence of instructions* to be carried out to solve the problem of making chocolate chip cookies
  - It begins with a list of ingredients which is the *input*
  - The sequence of instructions specify the actions to be carried out with the input to produce the cake which is the *output*
  - It takes a *finite time* to execute the instructions and stop

# A Wrong Algorithm

- What is wrong with this so called algorithm?
  - Directions: (From the back of a shampoo bottle.)
  - Wet hair.
  - Apply a small amount of shampoo, lather, rinse, repeat.
- If you follow this algorithm, you will never finish washing your hair!



# Properties of Algorithms (1/2)

- **Inputs:**
  - A number of quantities are provided to an algorithm initially before the algorithm begins. These quantities are the *inputs* which are processed by the algorithm.
- **Definiteness:**
  - The *processing rules* specified in the algorithm must be precise, unambiguous and lead to a specific action.
- **Effectiveness:**
  - Each instruction must be sufficiently basic such that it can, in principle, be carried out in a finite time by a person with paper and pencil.

# Properties of Algorithms (2/2)

- **Finiteness:**

- The total time to carry out all the steps in the algorithm must be finite. An algorithm may contain instructions to repetitively carry out a group of instructions. This requirement implies that the number repetitions must be finite

- **Outputs:**

- An algorithm must have one or more output

- **Correctness:**

- Correct set of output must be produced from the set of inputs. For the same input data, it must always produce the same output

# Origin of word Algorithm

- Named after famous Persian mathematician Abu Jafar Mohammad ibn Musa al-Khowarizmi, who wrote the book “Kitab al jabr Walmuqabla” (Rules of restoration and reduction) around 825 A.D.
  - The last part of his name get corrupted to algorithm
- The word algorism originally referred only to the rules of performing arithmetic using Arabic numerals but evolved into algorithm by the 18th century
- The first case of an algorithm written for a computer was Ada Byron’s notes on the analytical engine written in 1842, for which she is considered by many to be the world’s first programmer

# Writing Algorithms

- There are various techniques that can be used to write algorithms:
  - in English (e.g. enrolment guide)
  - by diagrams (e.g. instructions for assembling kitset furniture)
  - by Flowcharts
  - in pseudocode
  - in a computer programming language
  - in some other more-or-less formal language (e.g. knitting pattern)

# A Simple Example

- Fahrenheit to Celsius Conversion
  - **Step 1:** Read the temperature in Fahrenheit from the keyboard
  - **Step 2:** Calculate its Celsius equivalent as  $(F-32) \times 5/9$ , where F is the inputted Fahrenheit value
  - **Step 3:** Print the calculated Celsius value on the screen

# Another Example (1/2)

- *An algorithm to pick the largest tender from a set of tenders*
- Step 1: Read the first tender and note down its value as the maximum tender value so far encountered. Note down the tender identification number
- Step 2: As long as tenders are not exhausted do Steps 3 and 4. Go to Step 5 when tenders are exhausted
- Step 3: Read the next tender and compare the tender with the current maximum tender value

# Another Example (2/2)

- Step 4: If this tender value is greater than that previously noted down as maximum tender then erase the previous maximum value noted and replace it by this new value. Replace the previously noted tender identification number by this identification number. Else do not do anything
- Step 5: Print the final value of maximum tender and its identification noted down in Step 4

# Pseudocode

- Pseudocode is a written statement of an algorithm using a restricted and well-defined vocabulary
- It is, as the name suggests, cannot be executed on a real computer, but models and resembles real programming code, and is written at roughly the same level of detail
- It is very much like a 3GL and is therefore more easily translated into programming language
- For many programmers and program designers is the preferred way to state algorithms and program specifications
- No standard rules for writing



# Writing Algorithms in Pseudocode

- Pseudocode contains two main elements, the first being keywords
- These keywords often resemble programming commands and often occur as pairs of words
  - When writing a keyword in pseudocode we would often write it in UPPERCASE and may even highlight it using **boldface**
- The second element is that pseudocode relies on indenting to show structure
  - This enables the person working with the algorithm to easily see the flow of logic through the algorithm

# Example

- *Computing area of a rectangle*
- Input: length and breadth of the rectangle
- Output: Area of the rectangle
- Step 1. READ length
- Step 2. READ breadth
- Step 3. COMPUTE length \* breadth GIVING area
- Step 4. PRINT area

# Values and Variables

- Values

- Represent quantities, amounts or measurements
- May be numerical or alphabetical (or other things)
- Often have a unit related to their purpose
- Example:  
Recipe ingredients



This jar  
can contain

*Values*

10 cookies  
50 grams of sugar  
3 slices of cake  
*etc.*

- Variables

- Are containers for values
- places to store values

# Components of an Algorithm

- Values and Variables
- Instruction (a.k.a. primitive)
  - Some action that is simple and unambiguous
- Sequence (of instructions)
  - A series of instructions to be carried out one after the other
- Selection (between instructions)
  - An instruction that decides which of two possible sequences is executed
  - The decision is based on a single true/false condition
- Repetition (of instructions)
  - Repeat an instruction. while (or maybe until) some true or false condition occurs
  - Test the condition each time before repeating the instruction

# Writing Pseudocodes

<b>Group</b>	<b>Key words</b>	<b>Example</b>
Input/Output	<p><b>INPUT, READ</b> Used to get values from a data source, a keyboard for instance</p> <p><b>DISPLAY, PRINT</b> Used to output values to a data sink, a screen or printer for instance</p>	<p><b>INPUT</b> counter</p> <p><b>DISPLAY</b> new_value</p>
Processing	<p><b>ADD</b></p> <p><b>SUBTRACT</b></p> <p><b>COMPUTE</b></p> <p><b>SET</b></p>	<p><b>ADD 3 TO</b> count</p> <p><b>SUBTRACT 5 FROM</b> count</p> <p><b>SET</b> count <b>TO</b> 12</p> <p><b>COMPUTE</b> 10 + count</p> <p><b>GIVING</b> new_count</p>

# Writing Pseudocodes

<b>Group</b>	<b>Keywords</b>	<b>Example</b>
Repetition	REPEAT statement UNTIL <condition>  DOWHILE <condition> statement END DOWHILE  FOR <var> = <start value> to <stop value> ENDFOR	SET count_value TO 0 REPEAT DISPLAY count_value ADD 1 TO count_value; UNTIL count_value > 10  DOWHILE count_value < 10 DISPLAY count_value count_value = count_value + 1 END DOWHILE  FOR count = 1 to 10 DISPLAY count + count ENDFOR

# Writing Pseudocodes

Group	Keywords	Example
Decision	<pre>IF &lt;condition&gt; THEN     statement ENDIF  IF &lt;condition&gt; THEN     statement ELSE     statement ENDIF</pre>	<pre>IF count &gt; 10 THEN     DISPLAY count ENDIF  IF count &gt; 10 THEN     DISPLAY 'count &gt; 10'     ADD 4 to sum ELSE     DISPLAY 'count &lt;= 10'     ADD 3 to sum ENDIF</pre>

# Using Operators

- Although the processing group are very useful most program designers tend to prefer to use operators like:

- +      add
- -      subtract
- \*      multiply
- /      divide
- =      assign

- These are the *arithmetic* operators

- These are more intuitive since we use them in arithmetical expressions and they are easier to write, for instance:

- count = count + 22
- x = count / 12
- sum = sum - count
- x = count \* sum



# Relational Operators

Operator	Example	Description
== equal to	IF count == 32 THEN ...	This decision states that if the variable <i>count</i> contains 32 execute the statement following THEN.
<= less than or equal to	DOWHILE count <= 50 ... END DOWHILE	The statements in the while loop will execute while the value of <i>count</i> is 50 or less.
>= greater than or equal to	REPEAT ... UNTIL count > 12	The statements in the repeat loop will execute until the value of <i>count</i> exceeds 12.
<> not equal to	IF count <> total THEN ...	If the value of <i>count</i> is not the same as the value of <i>total</i> execute the statement following THEN.

# Logical Operators

Operator	Example	Description
AND	IF (x == 32) AND (y == 7) THEN sumxy = x + y	if it is true that $x == 32$ AND it is also true that $y == 7$ then add x and y together and store the result in variable sumxy.
OR	IF (letter == 'A') OR (letter == 'E') THEN DISPLAY 'Vowel'	if the variable <i>letter</i> contains either 'A' or 'B' then display the word vowel on the output device
NOT	IF NOT (letter == 'A') THEN DISPLAY 'Not letter A'	If it is true that letter is equal to 'A', the NOT operator in <i>NOT (letter = 'A')</i> will give a value of FALSE.

# Example 1

- *Finding largest of two numbers*
- 1. READ num1
- 2. READ num2
- 3. IF num1 > num2 THEN
- 4.     PRINT 'num1 is larger'
- 5. ELSE
- 6.     PRINT 'num2 is larger'
- 8. ENDIF

# Example 2

- *Sum of first n natural numbers*
- 1. READ n
- 2. counter = 1
- 3. sum = 0
- 4. DOWHILE counter <= n
- 5        sum = sum + counter
- 6.        counter = counter + 1
- 7. END DOWHILE
- 8. DISPLAY sum

# Developing an Algorithm

- Understand the problem (Do problem by hand. Note the steps)
- Devise a plan (look for familiarity and patterns)
- Carry out the plan (trace)
- Review the plan (refinement)

# Problem Example (1/5)

- *Find the average from a given set of numbers*
- **1. Understanding the problem:**
  - (i) Write down some numbers on paper and find the average manually, noting each step carefully. e.g. Given a list say: 5, 3, 25, 0, 9
  - (ii) Count numbers, i.e. How many? 5
  - (iii) Add them up, i.e.  $5 + 3 + 25 + 0 + 9 = 42$
  - (iv) Divide result by numbers counted, i.e.  $42/5 = 8.4$

# Problem Example (2/5)

- **2. Devising a plan:**

- Make note of not what you did in steps (i) through (iv), but how you did it. In doing so, you will begin to develop the algorithm.
- e.g. How do we count the numbers?
  - Starting at 0 i.e. set **COUNTER** to 0
  - Look at 1st number, add 1 to **COUNTER**
  - Look at 2nd number, add 1 to **COUNTER**
  - and so on,
  - until you reach the end of the list

# Problem Example (3/5)

- How do we add numbers?
  - Let **SUM** be the sum of numbers in list. Set **SUM** to 0
  - Look at 1st number, add number to **SUM**
  - Look at 2nd number, add number to **SUM**
  - and so on,
  - until we reach end of list
- How do we compute the average?
  - Let **AVE** be the average
  - then **AVE** = total sum of items/ number of items, i.e. **SUM/COUNTER**



# Problem Example (4/5)

- **3. Identifying patterns, repetitions and familiar tasks.**
  - Familiarity: Unknown number of items? i.e.  $n$  item
  - Patterns : look at each number in the list
  - Repetitions: Look at a number
    - Add number to sum
    - Add 1 to counter

# Problem Example (5/5)

- **4. Carrying out the plan**
  - Check each step
  - Consider special cases
  - Check result
  - Check boundary conditions:
    - i.e. what if the list is empty?
    - Division by 0?
    - Are all numbers within the specified range?
- In this example, no range is specified - No special cases.
- Check result by tracing the algorithm with a list of numbers e.g. 7, 12, 1, 5, 13




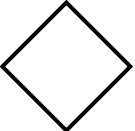
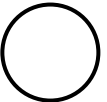

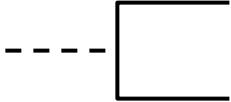
# Complete Pseudocode

- 1. total = 0
- 2. average = 0
- 3. count = 0
- 4. READ n
- 5. DOWHILE count < n
- 6.     READ number
- 7.     total = total + number
- 8.     count = count + 1
- 9. END DOWHILE
- 10. average = total/count
- 11. PRINT average

# Flowcharts

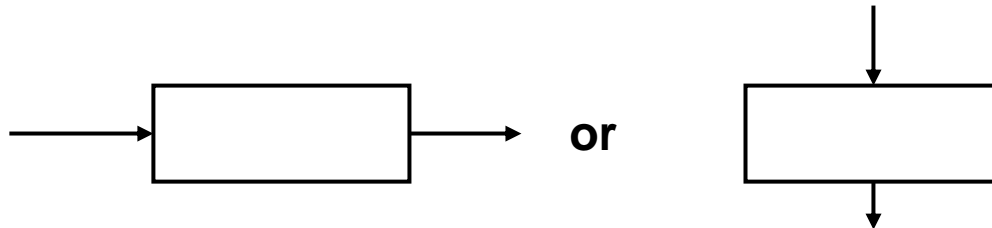
- Graphical representation of an algorithm that illustrates the sequence of operations to be performed using special-purpose symbols connected by arrows
- Flowcharts typically show a program's logic
- Generally drawn in the early stages of formulating computer solution
- Facilitate communication between programmers and business people
- These flowcharts play a vital role in the programming of a problem and are quite helpful in understanding the logic of complicated and lengthy problems

# Flowchart Symbols

	Start or end of the program
	Computational steps or processing function of a program
	Input or output operation
	Decision making and branching
	Connector or joining of two parts of program
	Flow lines
	Annotations

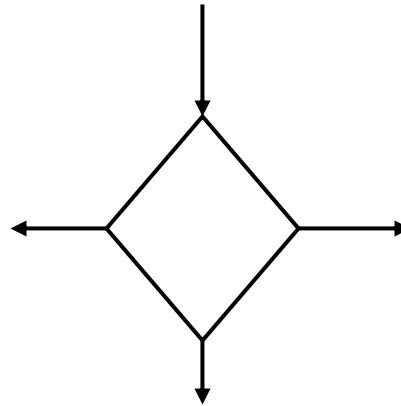
# Guidelines

- In drawing a proper flowchart, all necessary requirements should be listed out in logical order.
- The flowchart should be clear, neat and easy to follow There should not be any room for ambiguity in understanding the flowchart.
- The usual direction of the flow of a procedure or system is from left to right or top to bottom.
- Only one flow line should come out from a process symbol

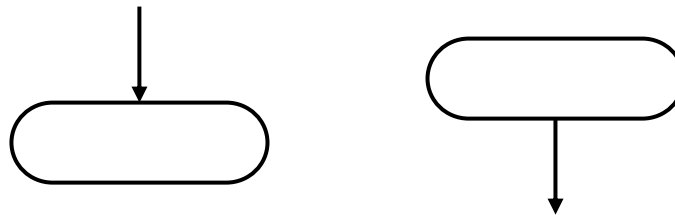


# Guidelines

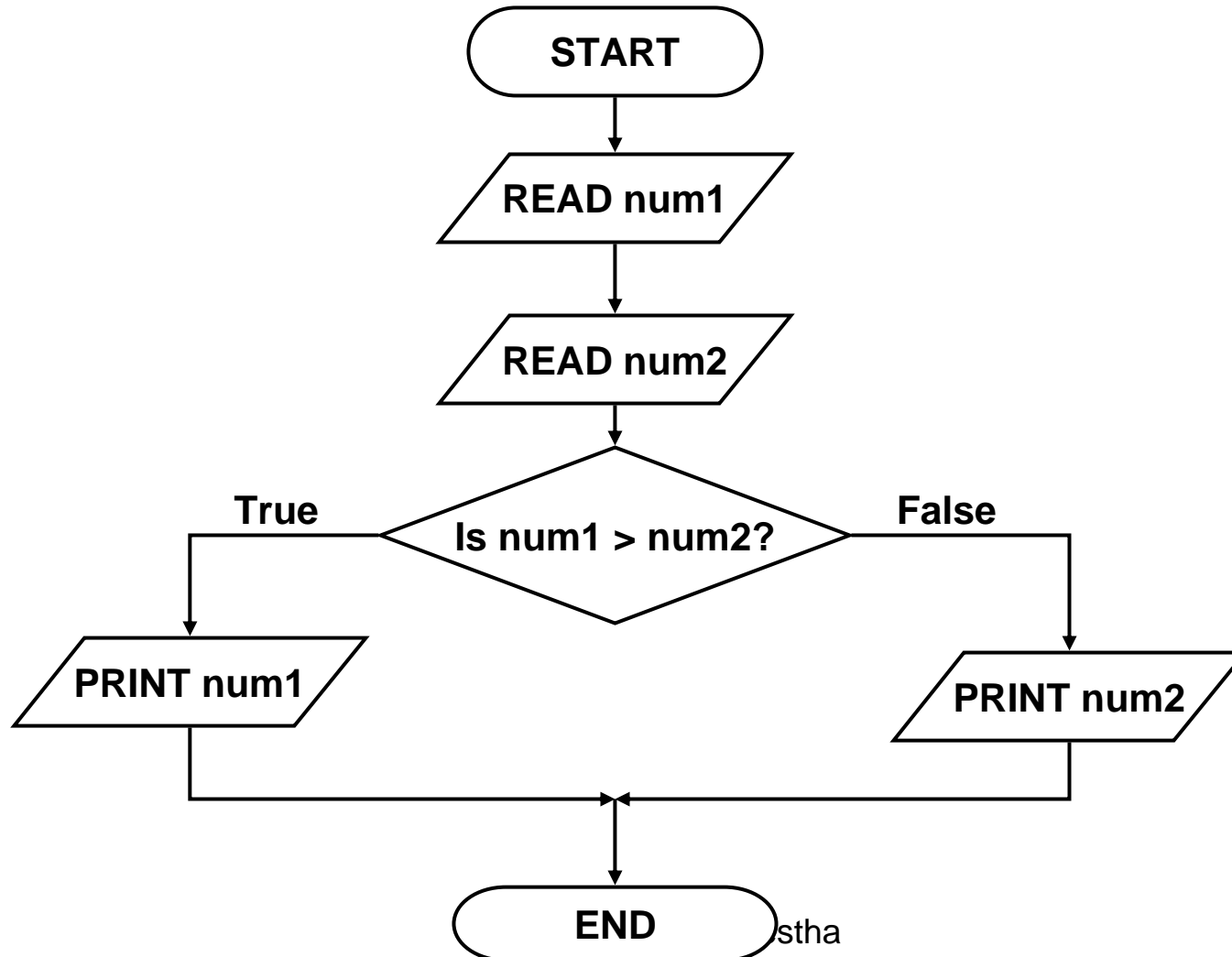
- Only one flow line should enter a decision symbol, but two or three flow lines, one for each possible answer, should leave the decision symbol



- Only one flow line is used in conjunction with terminal symbol

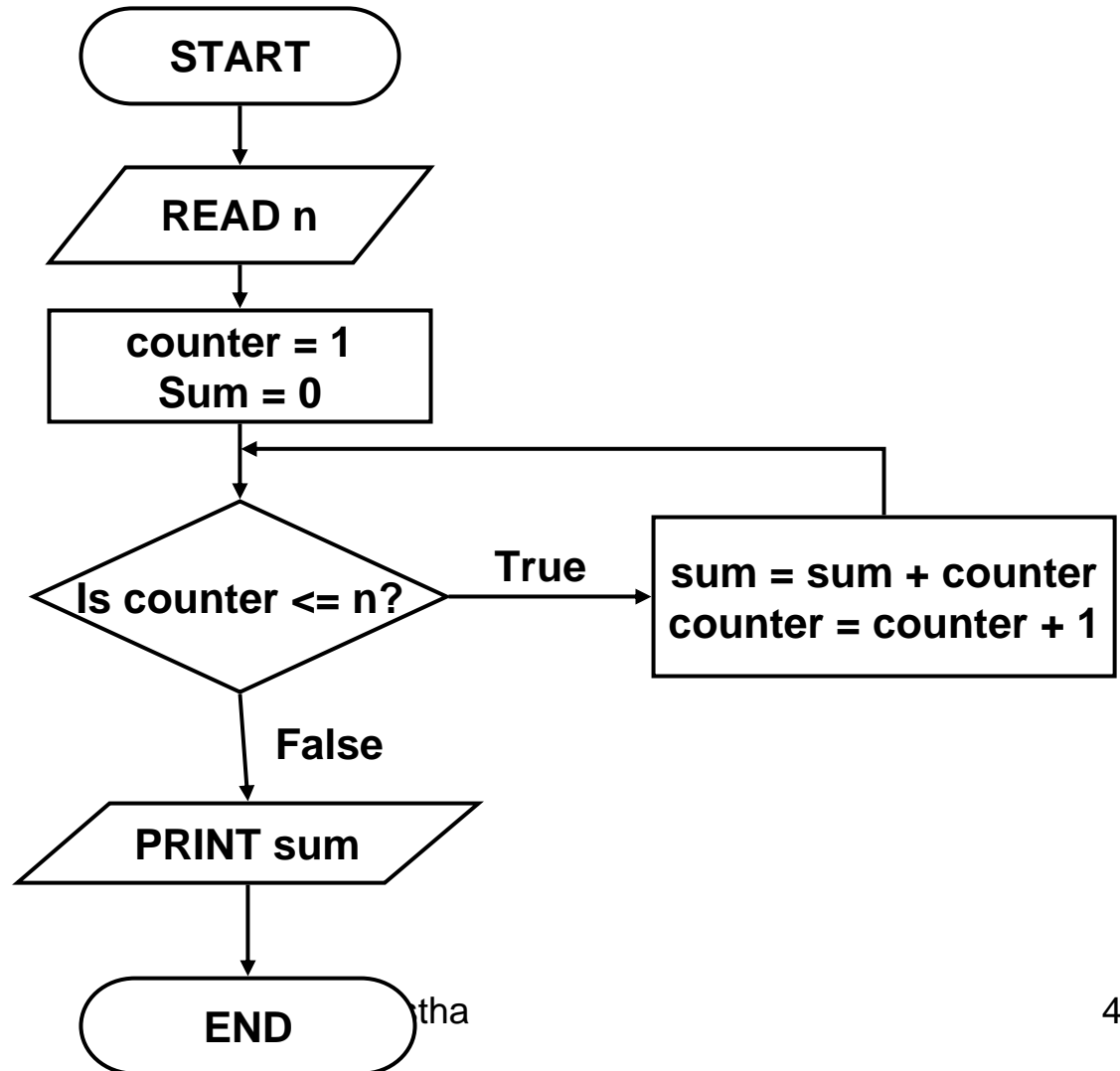


# Example 1: Larger of two numbers





# Example 2: Sum of first n natural numbers



# Some rules for flow charts

- Every flow chart has a START symbol and a STOP symbol
- The flow of sequence is generally from the top of the page to the bottom of the page. This can vary with loops which need to flow back to an entry point.
- Use arrow-heads on connectors where flow direction may not be obvious.
- There is only one flow chart per page
- A page should have a page number and a title
- A flow chart on one page should not break and jump to another page
- A flow chart should have no more than around 15 symbols (not including START and STOP)

# Advantages of Flowcharts

- ***Communication:*** Flowcharts are better way of communicating the logic of a system to all concerned
- ***Effective analysis:*** With the help of flowchart, problem can be analyzed in more effective way
- ***Proper documentation:*** Program flowcharts serve as a good program documentation, which is needed for various purposes
- ***Efficient Coding:*** The flowcharts act as a guide or blueprint during the systems analysis and program development phase
- ***Proper Debugging:*** The flowchart helps in debugging process
- ***Efficient Program Maintenance:*** The maintenance of operating program becomes easy with the help of flowchart. It helps the programmer to put efforts more efficiently on that part

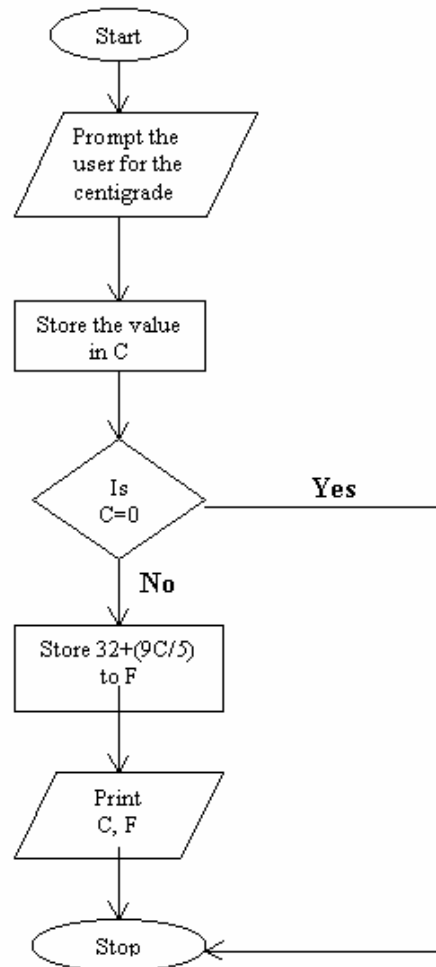
# Disadvantages of Flowcharts

- ***Complex logic:*** Sometimes, the program logic is quite complicated. In that case, flowchart becomes complex and clumsy.
- ***Alterations and Modifications:*** If alterations are required the flowchart may require re-drawing completely.
- ***Reproduction:*** As the flowchart symbols cannot be typed, reproduction of flowchart becomes a problem.
- The essentials of what is done can easily be lost in the technical details of how it is done

# System Flowchart

- System flowchart describes the data flow for a data processing system
- It provides a logical diagram of how the system operates. It represents the flow of documents, the operations performed in data processing system
- It also reflects the relationship between inputs, processing and outputs
- Following are the features of system flowcharts:
  - the sources from which data is generated and device used for this purpose
  - various processing steps involved
  - the intermediate and final output prepared and the devices used for their storage

# Example



# Program Flowchart

- A program flowchart represents, in detail, the various steps to be performed within the system for transforming the input into output
- The various steps are logical/ arithmetic operations, algorithms etc
- It serves as the basis for discussions and communication between the system analysts and the programmers
- Program flowcharts are quite helpful to programmers in organizing their programming efforts
- These flowcharts constitute an important component of documentation for an application

# Example

