Algorithms and Problem Solving

- Introduction
- What is an Algorithm?
- Algorithm Properties
- Example
- Exercises

What is an Algorithm?

What is an Algorithm?

- An algorithm is a precisely defined and ordered sequence of instructions that is guaranteed to solve a specific problem.
- The algorithm must be general, that is, it should solve the problem *for all possible input sets* to the problem.
- If there is a particular output for an input, we say that the algorithm can be *applied* to this input and *process* it to give the corresponding output.
- The word “algorithm” is derived from the name of Muhammad ibn Musa-al-Khwarizmi, a Muslim mathematician whose works introduced algebraic concepts.

Properties of Algorithms

Properties of Algorithm

- An algorithm should have the following properties:
  - No ambiguity in any instruction
  - No ambiguity which instruction is executed next
  - Finite number of instructions
  - Execution must halt and produce a result

Properties of Algorithm

- No ambiguity in any instruction: The instructions in an algorithm must be precisely defined, so that any person or machine implementing the instructions must have no difficulty in implementing them. For example, an instruction: "Adjust the temperature to suitable level" is ambiguous and not precisely defined.

Properties of an Algorithm – Contd.

- No ambiguity which instruction is executed next: The order in which instructions in algorithm must be executed should also be well defined and not ambiguous. Again, we can say that the ordering should be such that any person or machine implementing the algorithm must have no difficulty in implementing them.

- Finite number of instructions: The number of instructions in an algorithm must be finite so that implementation by hand or by machine takes a finite amount of time.

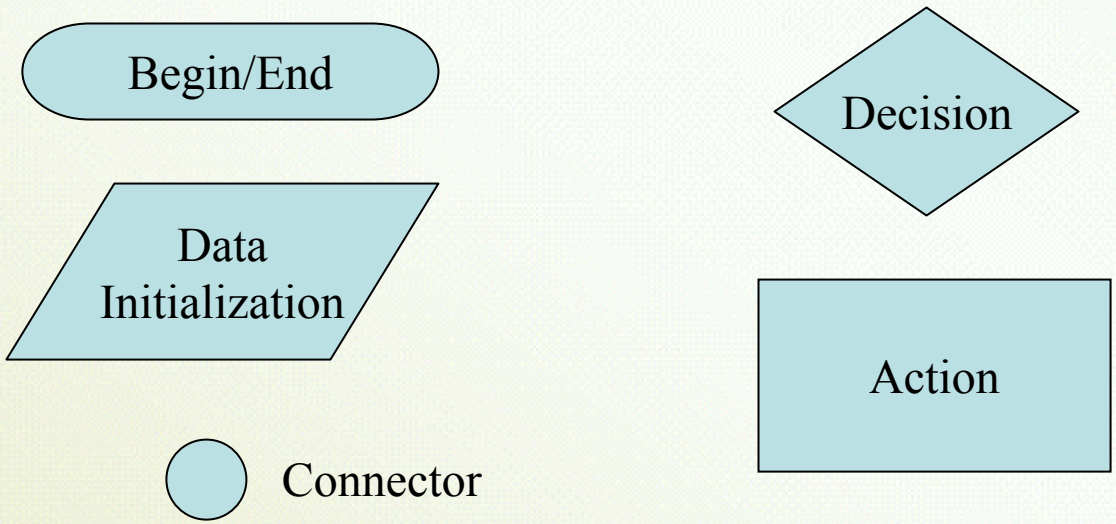
- Execution must halt and produce a result: Finally, an algorithm must halt and produce an output. Procedures that loop on forever on their inputs are not algorithms in the strict sense of the term.

Representation of Algorithms

- **Pseudocode** (*Pseudo = not real; false, Code = Program fragment*) is a generic way of describing an algorithm using the conventions of programming languages, without using language-specific syntax.
- Pseudocode generally does not actually use the syntax of any particular language
- There is no systematic standard form, although any particular writer will generally borrow the appearance of a particular language.
- Details not relevant to the algorithm (such as memory management code) are usually omitted.
- The programming language will be augmented with natural language where convenient (for example, for trivial operations such as swapping two variables).

Flow Charts

- Another way to represent algorithms is by using flowcharts.
- Flowcharts can be thought of as a graphical form of pseudocode.
- A flowchart (also spelled flow-chart and flow chart) is a schematic representation of a process.
- Generally the start point, end points, inputs, outputs, possible paths and the decisions that lead to these possible paths are included.
- Various flowchart boxes are shown below:



Example 1

- Develop an algorithm to find the average of a list of student scores.
- **Solution:**
- The problem statement doesn't specify the number of students, so we assume that a *sentinel value* of -1 for score indicates the end-of-input. The algorithm is straightforward.
- We keep inputting the score of each student.
- For each valid score entered we increment the variable count indicating the number of students.
- For each valid score entered, we add the score to another variable called sum.
- If a score is entered with a value of -1, it indicates the end of input so we stop inputting further scores.
- The average is equal to total/count.
- But what happens if the user didn't enter any valid score. For example his first entered value is -1?
- In that case if calculate total/count, it will lead to a division by zero error.
- So we say that if count is equal to zero, then the average is also zero.

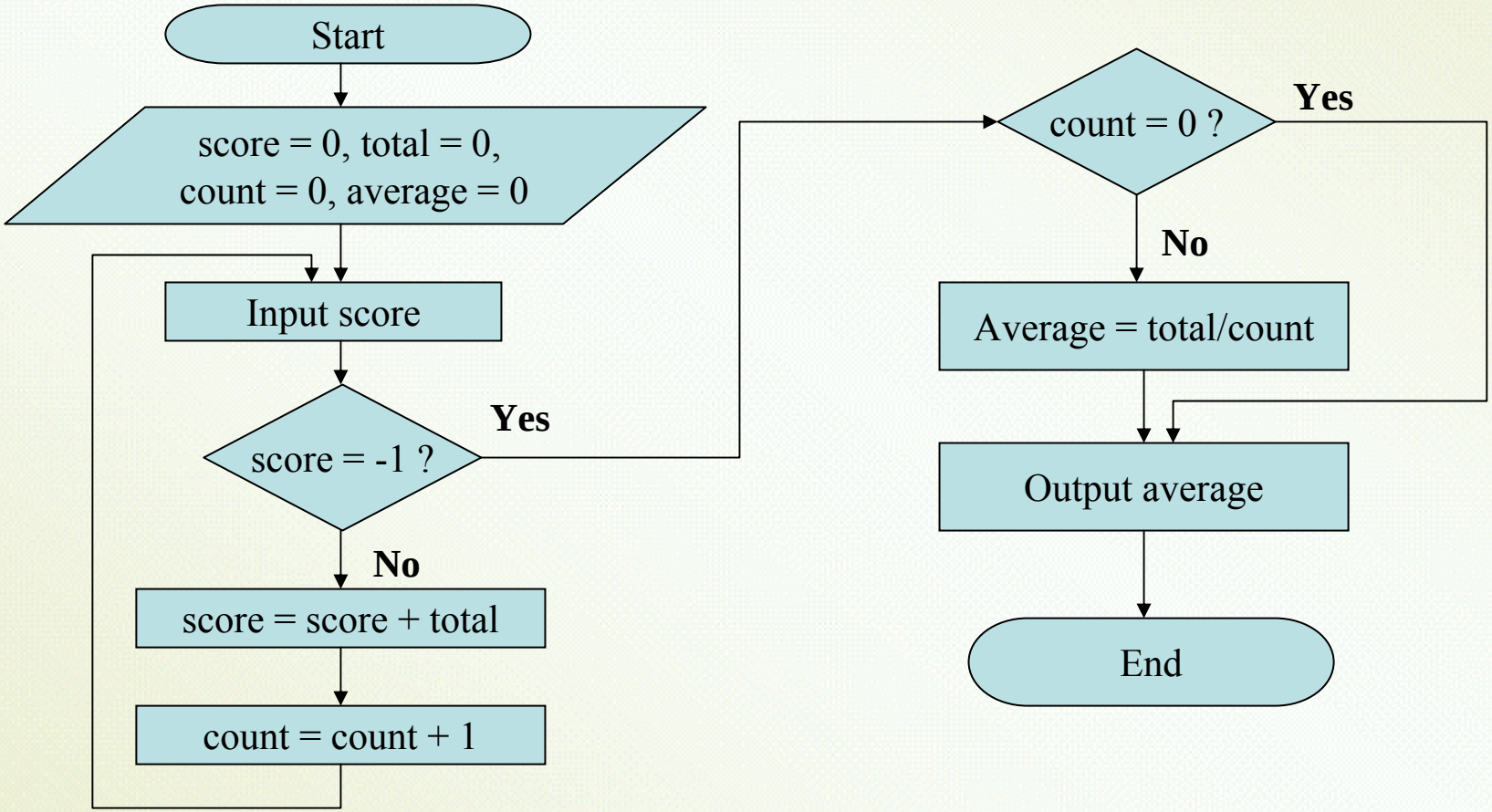
The Pseudocode version

- The algorithm is straightforward:

1. Initialize score, total, count and average to zero.
2. Input student score
3. If score = -1 then go to step 7.
4. Add score to total
5. Increment count by 1.
6. Go to step 2.
7. If count ≠ 0 average = total/count.
8. Output average.

Example 1 – Contd.

- The algorithm appearing in the previous slide was in pseudo-code that was informal. A flowchart version is shown below:



Example 1 - Conversion to Java Program

Below we show the Java program corresponding to Example 1. Observe the modifications made to the pseudocode.

```
import java.io.*;
public class Sentinel
{
    public static void main(String[] args) throws java.io.IOException
    {
        BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));
        double total = 0, avg = 0;
        int score = 0, count = 0;
        System.out.print("Enter the score (or -1 to Quit): ");
        score = Integer.parseInt(stdin.readLine());
        while (score != -1)
        {
            total += score;
            count += 1;
            System.out.print("Enter the score (or -1 to Quit): ");
            score=Integer.parseInt(stdin.readLine());
        }
        if (count != 0) avg = total/count;
        System.out.println("Average Grade = "+avg);
    }
}
```

Exercises

1. Develop an algorithm to check if a given integer is a perfect square or not. Give both the pseudocode version and the flowchart version. Convert your pseudocode into a Java program.

2. Develop an algorithm to display all prime numbers from 2 to 100. Give both the pseudocode version and the flowchart version. Convert your pseudocode into a Java program.

3. Develop an algorithm to find the maximum and minimum value from a list of integers. Give both the pseudocode version and the flowchart version. Convert your pseudocode into a Java program.