Problem Solving Concepts

By

Dr. Awad Khalil

Computer Science & Engineering Department

Problem Solving Methods

- There is no perfect method for solving all problems.
- There is no problem-solving computer to which we could simply describe a given problem and wait for it to provide the solution.
- Problem solving is a creative act and cannot be completely explained.
- However, we can still use certain accepted procedures to structure our thinking and help us solve problems.
- Three methods of problem solving are prominent:

1. Analytic Method.
2. Algorithmic Method.
3. Software Engineering Method.

Problem Solving Steps

- Each method has four basic components:

1. Problem: the problem presents the situation that requires a solution.
2. Reasoning: this implies a true comprehension of the problem.
3. Solution: this is the process that we develop to solve the problem.
4. Testing: this is the checking process we use to confirm that the solution is correct.

The Analytic Method

- Analytic problem solving has roots in mathematics, physics, engineering, chemistry, and a host of other areas in science and technology.
- Example

Problem

The sum of two numbers is s and their product is p. Find the 2 numbers.

Reasoning

Let us call the first number x and the second one y.

Therefore, $x + y = s$ and $x * y = p$.

From the first equation we get: $y = s - x$.

Substituting y from the first equation into the second equation, we get: $x * (s - x) = p$ or $x^2 - sx + p = 0$.

The Analytic Method

Solution

The solution to this problem is the solution to the quadratic equation: $x^2 - sx + p = 0$, provided that $s^2 - 4p \geq 0$.

This solution is known from the field of mathematics.

Testing

To check the correctness of the above solution we calculate $x + y$ and $x * y$.

The Analytic Method

- Limitations of the Analytic Approach
- Some problems cannot be solved analytically even with the most sophisticated technique. For example, mathematicians have proven that no analytic method can find the roots of a fifth-degree polynomial equation of the form:

$ax^5 + bx^4 + cx^3 + dx^2 + ex + f = 0$ for arbitrary coefficients.

- This claim does not contradict that in some specific cases, an analytic solution is possible. For example,

$x^5 - 1 = 0$ can be factored as $(x - 1)(x^4 + x^3 + x^2 + x + 1) = 0$ leading to one answer $x = 1$ that is easily verifiable.

- Likewise, there are numerous problems in the sciences and mathematics for which analytic solutions are not possible.
- Algorithms are often used to solve such problems and the answers that result are accurate enough to be accepted as solutions.

The Algorithmic Approach

Algorithmic Problem

An *algorithmic problem* is any problem whose solution can be expressed as a list of executable instructions. By executable we mean that an independent executor (human or machine) should be able to carry out the instructions in a step-by-step manner.

Examples of Algorithmic Problems

1. Backing a cake.
2. Knitting a sweater.
3. Taking the sum of a finite list of numbers.
4. Sorting a finite list of names.
5. Playing Tic-Tac-Toe.

Algorithm

- An algorithm is a sequence of executable instructions with these properties:

1. There is no ambiguity in any instruction.
2. There is no ambiguity about which instruction is to be executed next.
3. The description of the algorithm is finite.
4. Execution of the algorithm concludes after a finite number of steps.

- Algorithms are not unique to the computational sciences. An algorithm is a recipe, a sequence of steps that leads from a starting point to a finished product.

The Algorithmic Approach

- Non-Algorithmic Problems

Consider the following instructions:

Step1: Make a list of the odd positive integers.
Step2: Compute their sum.
Step3: Compute their average.
Step4: Print the average.

- This is not an algorithm because it is impossible to make an infinite list of numbers, and the problem of finding the sum (or average) of all odd positive integers is not algorithmic.

Phases of Algorithmic Problem Solving

- Ultimately, one would like to reduce the process of problem solving to an algorithm in itself, but this has been shown to be impossible.
- The following loosely defined problem-solving phases were presented by the mathematician G. Poyla in the late 1940s.

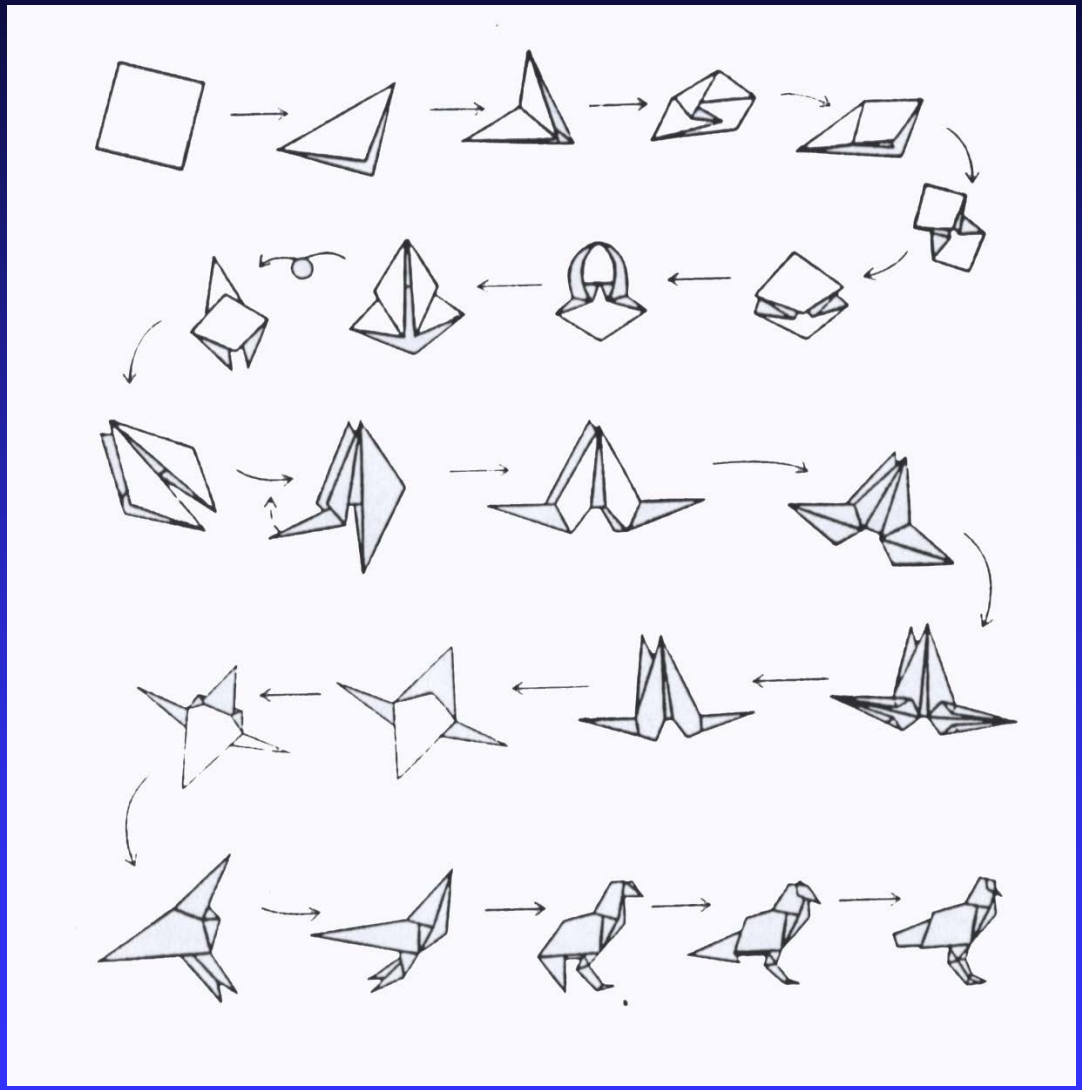
Phase1: Understand the problem.

Phase2: Devise a plan for solving the problem.

Phase3: Carry out the plan.

Phase4: Evaluate the solution for accuracy and for its potential as a tool for solving other problems.

An Algorithm



Software Engineering Method

- Software engineering
  - Area of computer science concerned with building large software systems
- Challenge
  - Tremendous advances in hardware have not been accompanied by comparable advances in software

Software Engineering Phases

- Problem Analysis - (Correct Problem)
  - Identify data objects
  - Goal to model properties
  - Determine Input / Output data
  - Constraints on the problem
- Design
  - Decompose into smaller problems
  - Top-down design (divide and conquer)
  - Develop Algorithm (Desk check)

Software Engineering Phases (Cont'd)

- Implementation
  - Writing the algorithm
- Testing
  - Verify the program meets requirements
  - System and Unit test
- Documentation
  - Key part in the development process

Software Engineering Goals

- Reliability
  - An unreliable life-critical system can be fatal
- Understandability
  - Future development becomes very difficult if software is hard to understand
- Cost Effectiveness
  - Cost to develop and maintain should not exceed profit

Software Engineering Goals (Cont'd)

- Adaptability
  - System that is adaptive is easier to alter and expand
- Reusability
  - Improves reliability and maintainability, and reduces development costs

Applying the Software Development Method

- Case Study: Converting Miles to Kilometers
  - Problem Your summer surveying job requires you to study some maps that give distances in kilometers and some that use miles. You and your coworkers prefer to deal in metric measurements. Write a program that performs the necessary conversion.

Applying the Software Development Method

- Analysis The first step in solving this problem is to determine what you are asked to do. You must convert from one system of measurement to another, but are you supposed to convert from kilometers to miles, or vice versa? The problem states that you prefer to deal in metric measurements, so you must convert distance measurements in miles to kilometers.

Applying the Software Development Method

- Design The next step is to formulate the algorithm that solves the problem. Begin by listing the three major steps, or sub problems, of the algorithm.
- Implementation To implement the solution, you must write the algorithm as a C++ program.
- Testing How do you know the sample run is correct?

Algorithmic Structure

- An algorithm is written as a step-by-step procedure (**sequence**) in which choices can be made where necessary (**selection**), and all or part of the algorithm can be repeated (**repetition**).

- Thus, the basic structures of an algorithm are:

  1- sequence

  2- selection

  3- repetition

Sequence

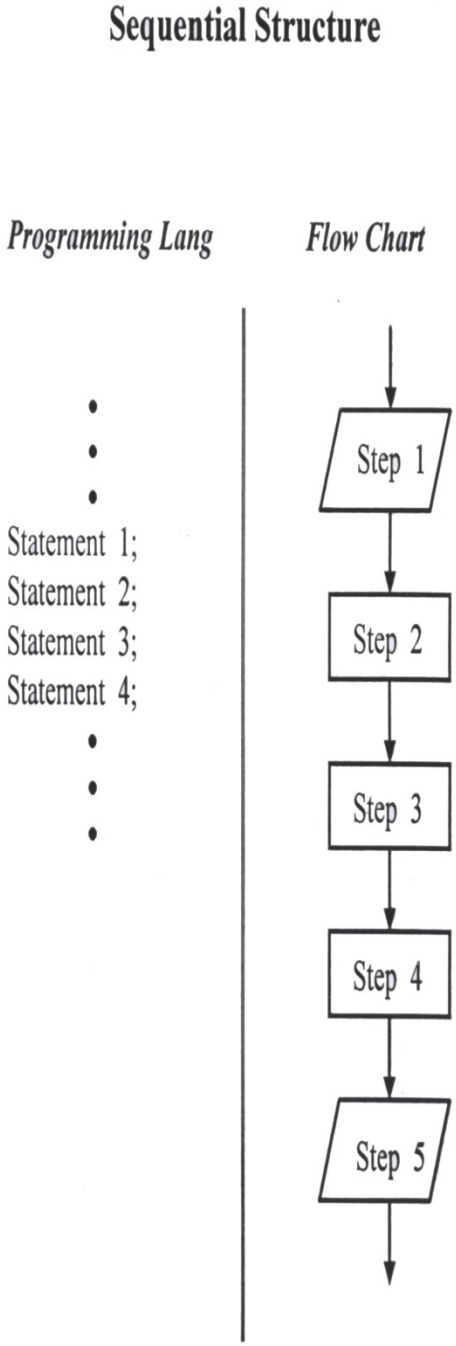
- An algorithm is based on the notion of sequence, which is the ordering of statements.
- Step *n* cannot be started until step *n*-1 is complete.
- *Problem1*: Develop an algorithm that calculates and prints the area of a trapezoid when the values of the bases and height are given as input.

Input: `base1, base2, height`

Output: `area`

Algorithm1:

```
step1: INPUT base1, base2, height
step2: area := (base1+base2)*height/2
step3: OUTPUT area
step4: STOP
```

Algorithm Representation

- There several methods to represent an algorithm:
  - Pseudocode
  - Flow Chart
  - Programming Languages

Algorithm Representation

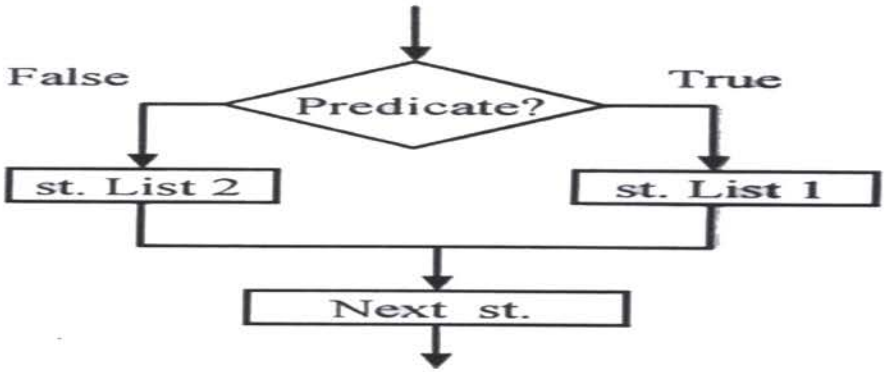
- The artificial language we are using to describe an algorithm is called **pseudocode**.
- Pseudocode is a notation that can be used to express ideas informally during the algorithm development process.
- The pseudocode used during the early stages of algorithm development resemble but is less formal than a programming language.
- *base1*, *base2*, *height*, and *area* represent the names of **variables** because their values can change depending on the actions taken in the algorithm.
- In a programming language, such as C++, the names of variables are called **identifiers** because they serve to identify, by name, the memory locations in the computer where the corresponding data are stored.
- Although the names can be chosen arbitrary, as a matter of style it is better to choose meaningful identifiers that suggest what the name means.

Selection

- Selection is the choice of alternate paths (branches) depending on a condition that may arise in the logical flow of the algorithm.

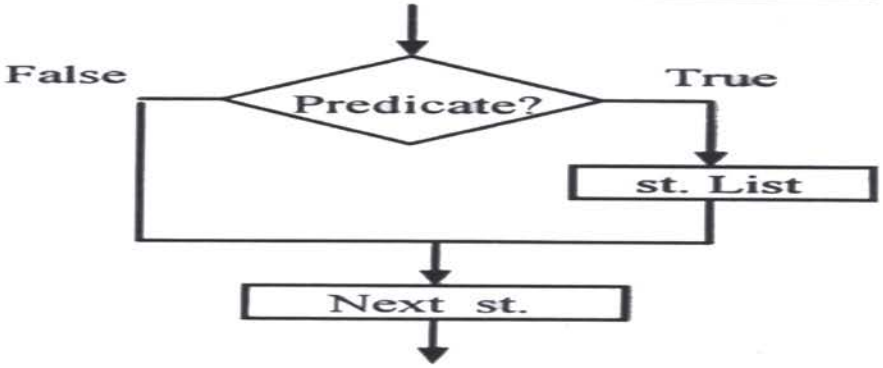
Selective Structure

```
if <Predicate> then
        <st. List 1>
else
        <st. List 2>
```



Two-side selective str.

```
if <Predicate> then
        <st. List>
```



One-side selective structure

Selection

- *Problem2*: Write an algorithm that accepts a number representing either a Fahrenheit or a Celsius temperature scale and converts it to the other scale.

Input: `temperature, scale` (Fahrenheit or Celsius)

Output: `newTemp` (Converted Temperature)

Algorithm2:

```
step1: INPUT scale, temperature
step2: IF scale = 'f' THEN
          newTemp := 5/9*(temperature - 32)
       ELSE
          newTemp := 9/5*temperature + 32
step3: OUTPUT newTemp
step4: STOP
```

- The above algorithm has introduced `IF...THEN...ELSE`, which is called a **selection structure**. This statement reads "*IF it is true that the scale equals f (Fahrenheit) THEN convert the input temperature to Celcius ELSE convert it to Fahrenheit*".
- In an `IF...THEN...ELSE` statement, if the logical condition after `IF` is true, the statements after `THEN` are executed; otherwise, the statements after `ELSE` are executed.

Selection

- *Problem3*: Assume that a salesperson is paid a commission based on the number of sales made during the week. The salesperson is paid LE 8 per sale for fewer than the established quota of 15 sales, LE 12 per sale if the quota is reached, and LE 16 per sale if the quota is exceeded.

Input: `sales` (i.e., number of sales)

Output: `commission`

Algorithm3:

```
step1: INPUT sales
step2: IF sales < quota THEN
          rate := 8
       ELSE IF sales = quota THEN
          rate := 12
       ELSE
          rate := 16
step3: commission := rate * sales
step4: OUTPUT commission
step5: STOP
```

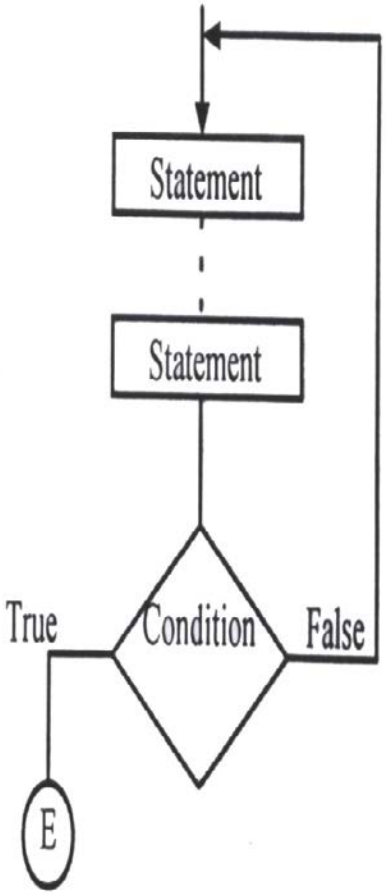
- Observe that `rate` must be calculated before the `commission` can be computed.
- This algorithm has introduced **multiple selection** based on the value of `sales`. Multiple selections can be achieved by the repeated use (or nested use) of `ELSE IF`.
- The last `ELSE` of the nested `IF` becomes the default or catchall case that is considered if none of the other possibilities is true.

Repetition

- The third tool used for the development of an algorithm is called **repetition** or **looping**, which provides for the repeated execution of part of the algorithm.

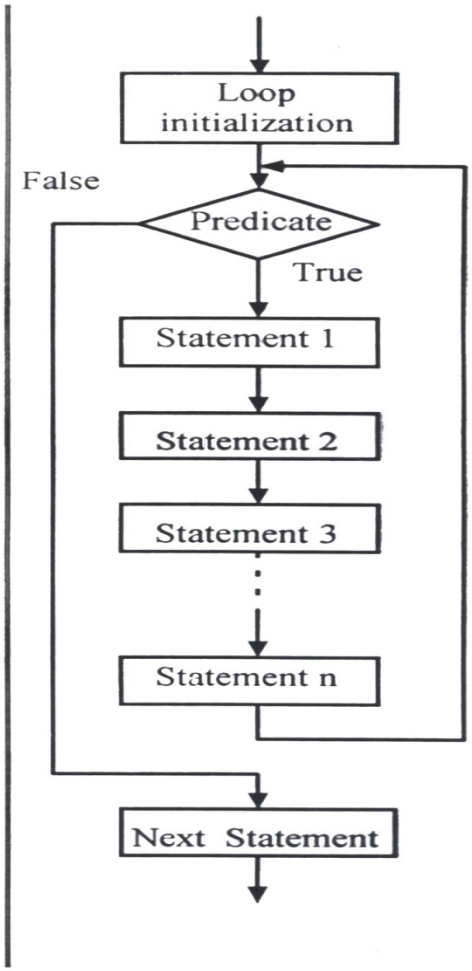
Using repeat statement

```
repeat
    <statement>;
    <statement>;
    .....
until <condition>;
```



Iterative Structure (Loops)

```
While <Predicate> do
    begin
      <Statement 1>;
      <Statement 2>;
      .....
      <Statement n>
    end;
```



Repetition

- *Problem4*: Reconsider the sales commission problem for many salespeople.

Input: number of sales people, number of sales for each salesperson.

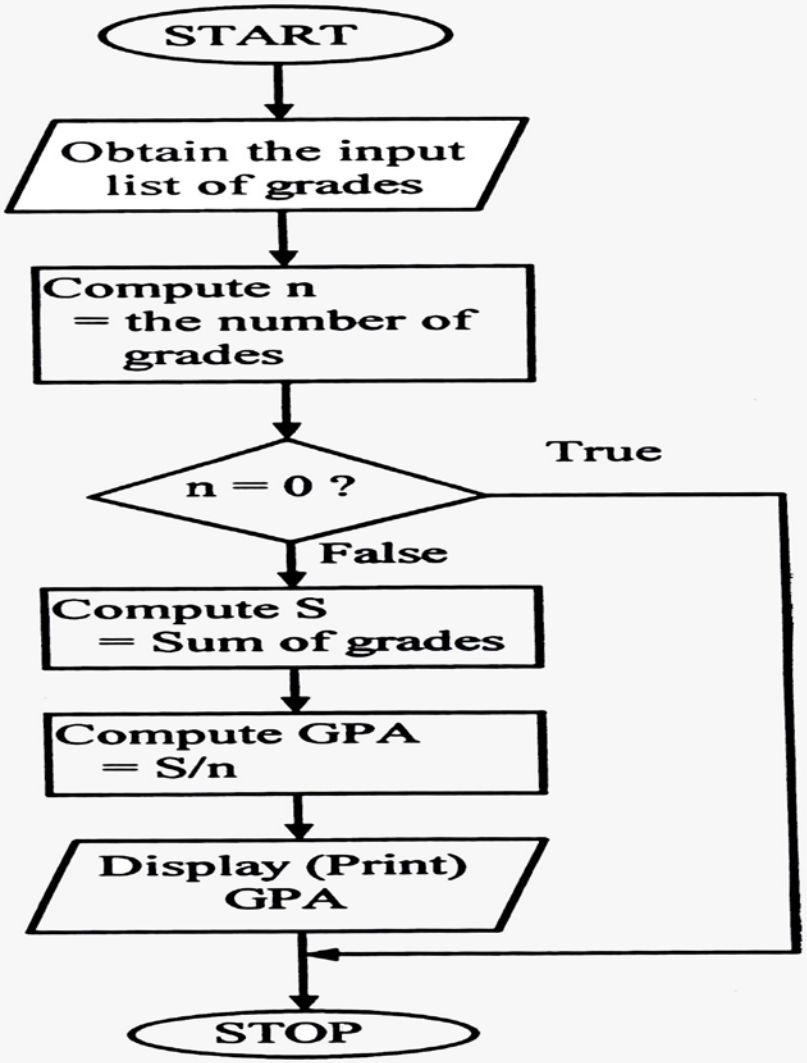
Output: commission for each salesperson.

Algorithm4:

```
step1:  INPUT numSalesPeople
step2:  LOOP numSalesPeople
   a:   INPUT sales
   b:   IF sales < quota THEN
                rate := 8
        ELSE IF sales = quota THEN
                rate := 12
        ELSE
                rate := 16
   c:   commission := rate * sales
   d:   OUTPUT commission
step3:  STOP
```

- A loop is used to repeatedly read the data for each salesperson and compute the associated commission.
- The word `LOOP` here means that the following sequence of steps (2a to 2d in this example) is to be repeated for a specific number of iterations (`numSalesPeople` in this example).
- The `LOOP` construct is only appropriate if the number of iterations can be predetermined as it is in the above algorithm.
- For many situations, however, it is not possible to predetermine the number of iterations and a more flexible looping structure is required.

Repetition


START
Obtain the input list of grades
Compute n = the number of grades
n = 0 ?
True
False
Compute S = Sum of grades
Compute GPA = S/n
Display (Print) GPA
STOP