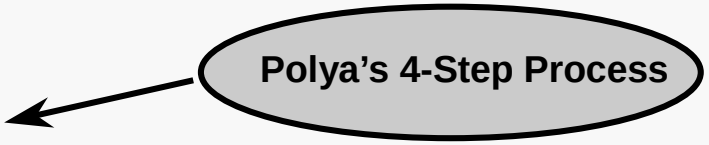
The Programming Process

Two-Phase Process



- Problem-Solving Phase
  1. Analysis & Specification
  2. General Solution (Algorithm)
  3. Verification.
- Implementation Phase
  1. Specific Solution (Program)
  2. Test

Maintenance Phase

1. Use (Install, Execute)
2. Maintain (Modifiy)

Polya's 4 Step Process

1. Understand the problem
    a. Know the boundaries of the problem
    b. Know the constraints on the solution
    c. Know what actions are allowed

2. Devise a plan
    a. Organize thoughts to develop a detailed Algorithm
    b. Use tools such as: outlining, flowcharting, and pseudocode.

3. Implement the plan
    a. Carry out the steps in the algorithm
    b. Translate the problem into a language understandable by the device to be used.

4. Test the plan
    a. Did the solution yield appropriate results?
    b. Can the solution be improved?

Understand the Problem

What are the data?

What is the required result?

What is the starting point (initial conditions)?

Is it at all possible to get the result from the data?

† Do the Problem by Hand.

What steps did you take? Can you write them down?

Is the problem divided into major parts? Can they be identified?

Have any problem assumptions been made? What are they?

Is the solution general?

Devise a Plan

Have you seen a similar problem before?

Do you know of a related problem with a useful solution?

- Can you use part of the related problem?
- Can the solution of the related problem be modified and used?

Look at the data... repetitions hint at loops in the solution.

- Identify the patterns in the data.

If you can't solve the problem, can you solve part of the problem?

Implement the Plan

Check each step of the plan.

Have you considered all the special cases?

Can you arrive at a reasonable result given...

- reasonable data?
- unreasonable data?

Make certain your solution works for "boundary conditions".

- Empty data? Data set too large?
- Data sets that results in illegal operations (division by zero)?

Test the Plan

Did you compute any intermediate results that were not used later?

Can they be eliminated?

Can the result be derived differently?

Can the solution be made simpler or more general?

Can the solution method be used for other problems?

Algorithm

- "A finite set of instructions that specify a sequence of operations to be carried out in order to solve a specific problem or class of problems." [Zwass]

- Properties of algorithms:
  - † Finiteness: Algorithm must complete after a finite number of instructions have been executed.
  - † Absence of Ambiguity: Each step must be clearly defined, having only one interpretation.
  - † Definition of Sequence: Each step must have a unique defined preceding & succeeding step. The first step (start step) & last step (halt step) must be clearly noted.
  - † Input: 0 or more predetermined number of data.
    Output: 1 or more results.
  - † Effectiveness: All instructions must be able to be performed. Illegal operations (division by 0) are not allowed.

Language Levels

Machine Language: (very low-level language)

- consists of strings of 0's and 1's (bits), binary coded
- different for each machine (machine dependent)
- all programs must be written in machine language (code) OR be translated into machine code before being executed

Assembly Language: (low or intermediate level)

- short abbreviations (mnemonics) are used for instructions; for the C++ statement: X = Y + Z, one might have (on an 8-bit machine):

Assembly language	Machine code
`LDA 9C2, A`	`00100110`
`ADD A, 9C#`	`00011010`
`STO A, 9C4`	`10000101`

Language Levels (cont)

High Level Languages:

- Logical and relatively English-like
- machine independent
- must be translated before execution (compiled or interpreted)

Some common examples:

COBOL	Common Business-Oriented Language
Fortran	Formula Translation
Pascal	designed for programming instruction
Ada	intended for general DOD contractor use
C	intended for system software development
C++	object-oriented extension of C language
Java	pure object-oriented design, portable but interpreted

Program Translation

- Source Code: high-level language instructions (C++)
- Object Code: low-level machine instructions