

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 and succeeding step. The first step (start step) and last step (halt step) must be clearly noted.
- Feasibility:** All instructions must be able to be performed. Illegal operations (division by 0) are not allowed.
- Input:** 0 or more data values.
- Output:** 1 or more results.

Algorithm: Newton-Raphson (3 iteration version)

1. Input: a real number X.
2. If (X < 0) Then
 Output: X " cannot be negative."
 STOP.
Endif
3. Set a variable, S, to 1.0.
4. Set a counter variable, I, to 3.
5. While (I > 0) Do
 - a. Calculate the value $(S + X / S) / 2.0$.
 - b. Set S to the value calculated in step 5a.
 - c. Subtract 1 from I.Endwhile
6. Output: "The square root of " X " is about " S
7. STOP.

Does this possess the properties of an algorithm listed on the previous slide?

There is no recipe for solving general problems. However these ideas are often useful:

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. Carry out 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. Review/extend the plan
 - a. Did the solution yield appropriate results?
 - b. Can the solution be improved?

George Pólya (1887-1985)

Do you understand all the words used in stating the problem?

What are you asked to find or show?

Can you restate the problem in your own words?

Can you think of a picture or a diagram that might help you understand the problem?

Is there enough information to enable you to find a solution?

Do you need to ask a question to get the answer?

Guess and check

Make an orderly list

Eliminate possibilities

Use symmetry

Consider special cases

Use direct reasoning

Solve an equation

Look for a pattern

Draw a picture

Solve a simpler problem

Use a model

Work backward

Use a formula

Be creative

Use your head/noggen

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)?

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?

Development Process

Problem-Solving Phase

1. Analysis & Specification
2. Design of a General Solution
3. Verification.

Implementation Phase

1. Production of a Specific Solution
 2. Test
-

Maintenance Phase

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

These are not really just a simple sequence of actions.

Discovered errors will force a re-examination of the algorithm, or perhaps the underlying analysis or even the problem specification.

Maintenance often costs more than development and marketing together. Especially if the design is ill-considered or the testing is inadequate.

Machine Language: (very low-level language)

- consists of strings of 0's and 1's (binary digits or bits); often expressed in base 16 (hexadecimal) for clarity
- different for each machine (machine dependent)
- all programs must be written in machine language (code) OR be translated into machine code before being executed

```
. . .  
8b 95 14 ff  
03 95 1c ff  
89 95 14 ff  
. . .
```

Assembly Language: (low or intermediate level)

- short abbreviations (mnemonics) are used for instructions; for the assembly code above, one might have:

```
. . .  
mov    edx, DWORD PTR _SumOfScores$[ebp]  
add    edx, DWORD PTR _nextScore$[ebp]  
mov    DWORD PTR _SumOfScores$[ebp], edx  
. . .
```

High Level Languages:

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

```
SumOfScores = SumOfScores + nextScore;
```

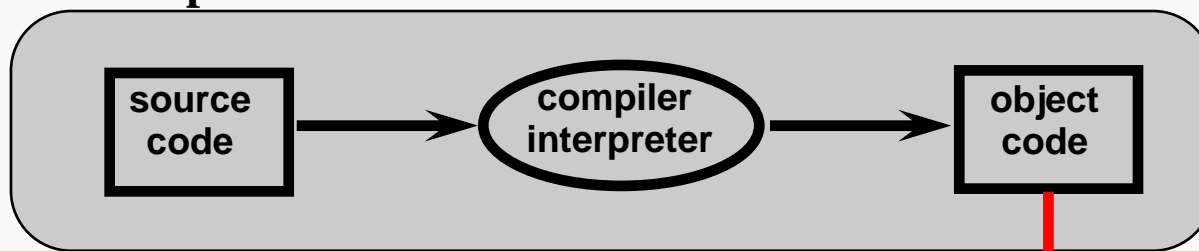
Some common high-level languages:

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

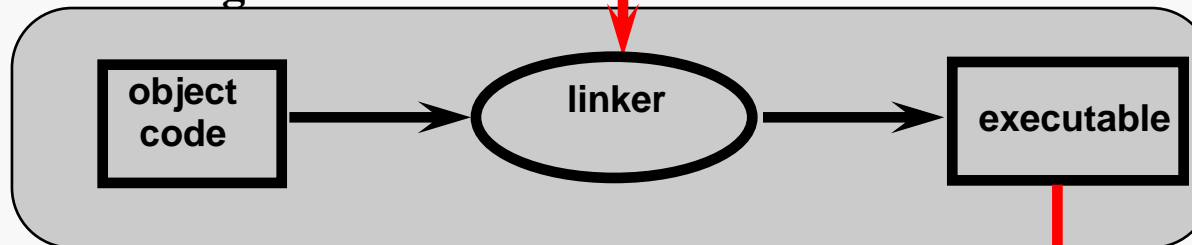
source code high-level language instructions (C++) written by the programmer

object code low-level machine instructions readable by the hardware

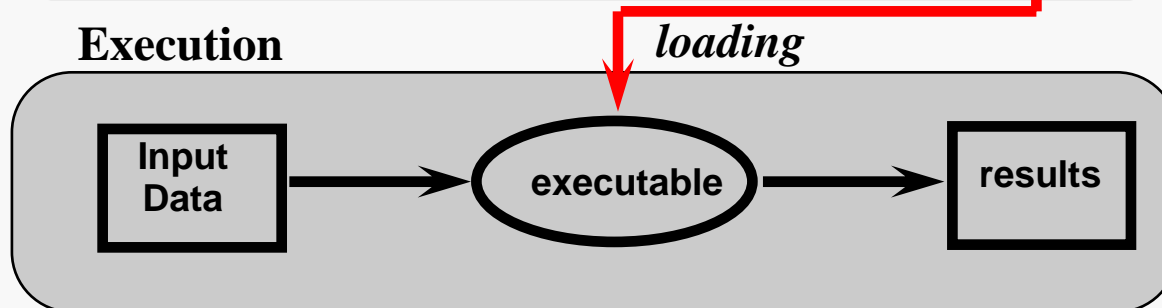
Compilation



Linking



Execution



Building

loading