ES117 LECTURE 3

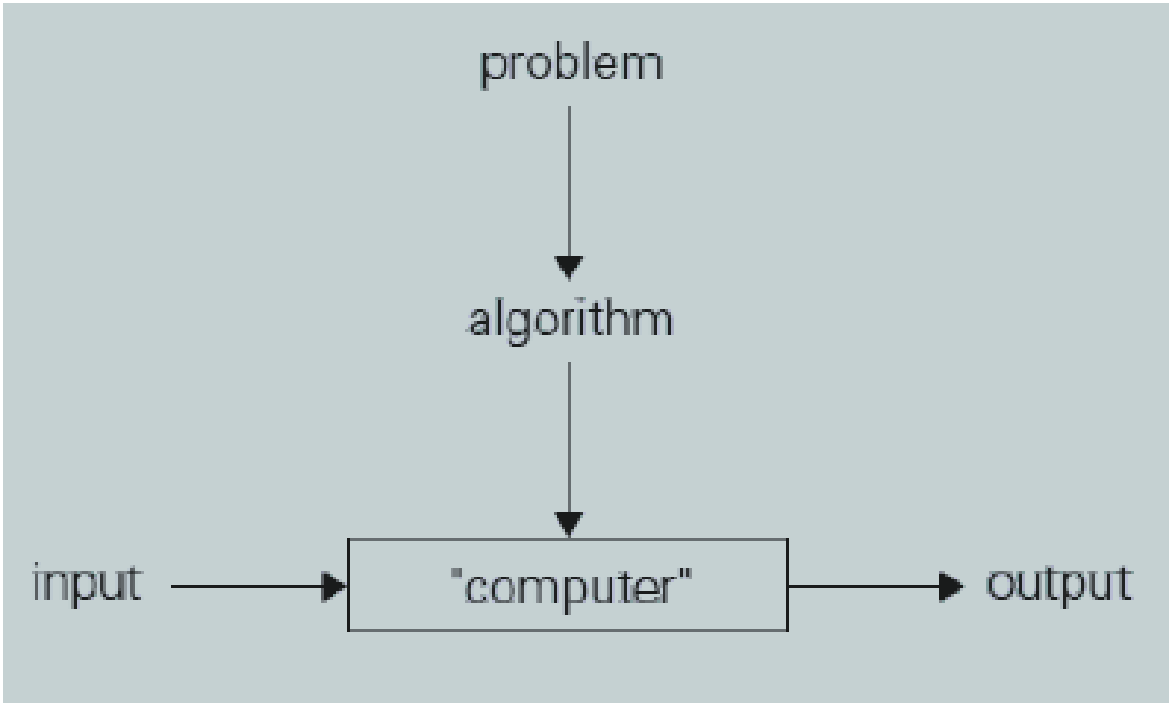
FUNDAMENTALS OF PROGRAMMING:

ALGORITHMS, PSEUDOCODES,

FLOW CHARTS, LANGUAGES

Introduction - Algorithms

We can consider algorithms to be practical solutions to problems. These solutions are not answers, but specific instructions for getting answers.



Introduction - Algorithms

* Before writing a program:
  * Have a thorough understanding of the problem
  * Carefully plan an approach for solving it
* While writing a program:
  * Know what "building blocks" are available
  * Use good programming principles

Algorithms

* Computing problems
  * All can be solved by executing a series of actions in a specific order
* Algorithm: procedure in terms of
  * Actions to be executed
  * The order in which these actions are to be executed
* Program control
  * Specify order in which statements are to be executed

Algorithms

* Decide the steps to be followed for solving the problem in question.
* Once you have designed an algorithm, you need to specify it in some fashion.
* Using a natural language is the obvious choice; however, the usual uncertainty of any natural language makes a brief and clear description of algorithms surprisingly difficult.

Algorithms – Pseudocode/Flowcharts

Pseudocode and Flowcharts

* Description of the algorithm in a standard form (easy to understand for anyone)
* Language independent

The constructs used to build algorithms can be described in two ways:

* Pseudocode (mix. of English and programming language)
* Flowcharts (using charts)

Algorithm – Pseudocode/Flowchart

Pseudocode

* Artificial, informal language that helps us develop algorithms
* Similar to everyday English
* Not actually executed on computers
* Helps us "think out" a program before writing it
  * Easy to convert into a corresponding program
  * Consists only of executable statements

Choice and decision-making – Pseudocode Example

alternatives in normal English language

```
if criterion 1
        action 1
else if criterion 2
        action 2
else if criterion 3
        action 3
otherwise
        action 4
```

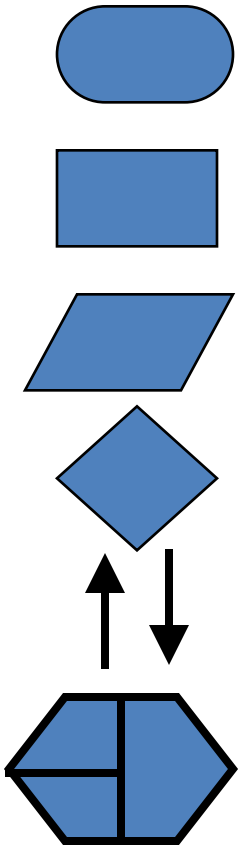
```
if (criterion 1)
        action_1;
else if (criterion 2)
        action_2;
else if (criterion 3)
        action_3;
else
        action_4;
```

Control Structures

* Flowchart
  * Graphical representation of an algorithm
  * Drawn using certain special-purpose symbols connected by arrows called flowlines
* Single-entry/single-exit control structures
  * Connect exit point of one control structure to entry point of the next (control-structure stacking)
  * Makes programs easy to build

Flowcharts

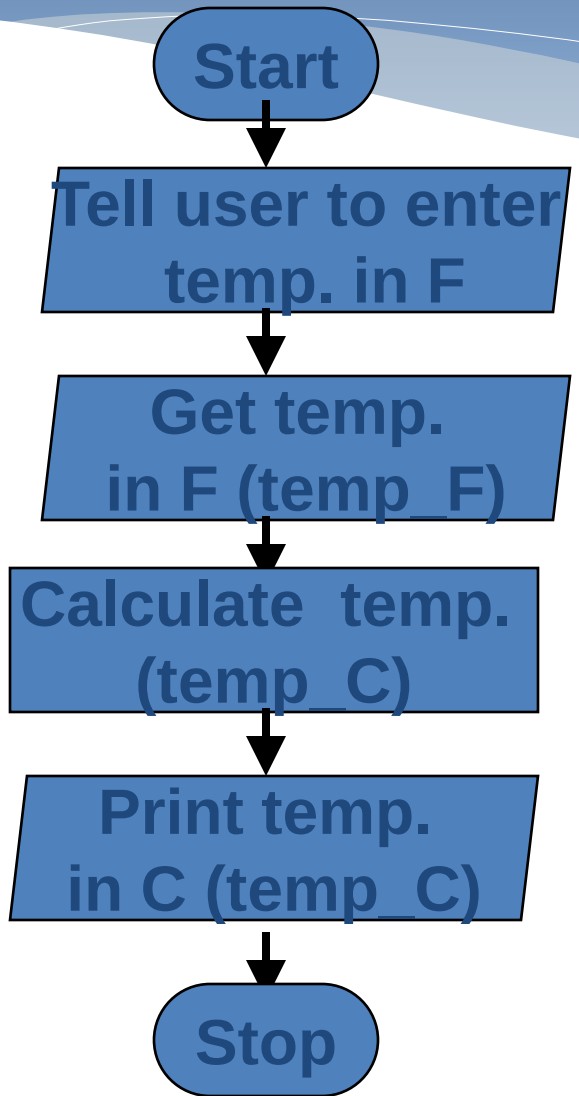
* Graphical representation of an algorithm
* Drawn using certain special-purpose symbols connected by arrows called flowlines



Start/ Stop
Acomputational result assigned to a var. at LHS
input / ouput operation
A point where a choice is made by two alternatives
Direction of program flow bw steps
İterative or counting loop

Flowcharts - Example

Temperature conversion from F to Celcius



temp_C=5/9*(temp_F- 32)

Coding an algorithm: Programming

* At the final step, the algorithm, expressed in either flowcharts or pseudocodes, are translated into the programming language (MATLAB, C, Fortran, etc.)

Programming

* The process of designing, writing, testing, debugging / troubleshooting, and maintaining the source code of computer programs
* Problem solving using a computer
* A program is a **sequence of instructions** that specifies how to perform a computation (mathematical or symbolic).
* The **source code** is written in programming language.
* Purpose is to create a program that exhibits a certain behavior.

Programming - Steps

Step 1: Problem analysis.

The designer must fully understand the nature of the problem to be solved and the need for the solution.

Step 2: Problem statement.

Develop a detailed statement of the problem to be solved with a computer program.

Step 3: Processing scheme.

Define the inputs required and the outputs to be produced by the program.

Programming - Steps

Step 4: Algorithm.

Design the step-by-step procedure using the top-down design process divide the overall problem into subordinate problems.

This list of tasks is the structure plan; it is written

in pseudocode, i.e. a combination of English, mathematics and estimated MATLAB commands or in a flowchart.

The goal is to design a plan that is understandable and easily translated into a computer language.

Programming - Steps

Step 5: Program algorithm.

Translate or convert the algorithm into a computer language (e.g. MATLAB) and debug the syntax errors until the tool performs successfully.

Step 6: Evaluation.

Test all of the options and conduct a validation study of

the computer program, compare results

* with other programs that do similar tasks,
* experimental data if appropriate
* theoretical predictions.

Step 7: Application.

Solve the problems using the program was designed to solve.

Good Programming Practice

A good program is:

* Efficient
* Correct (correct syntax & semantics)
* Readable (Easy to read & understand)
* Easy to maintain & enhance

Programming Languages

Computer languages are used to develop programs to be executed. They can be described in terms of levels.

Machine language:

- Lowest-level language
- Binary-encoded instructions that are directly executed by the hardware.
- Language is closely tied to the hardware design.
- Machine specific: machine language for one computer is different from that of a computer having a different hardware design.
- Strings of numbers giving machine specific instructions

Example:

+1300042774
+1400593419
+1200274027

Computing Software - Computer Languages

Assembly language:

- Also unique to a specific computer design.
- Commands or instructions are written in text statements instead of numbers.
- Assembly language programming requires good knowledge of the specific computer hardware.
- System software called an **assembler** translates the assembly language program to a machine language program for execution on the hardware.
- English-like abbreviations representing elementary computer operations (translated via assemblers)

Example:

```
LOAD   BASEPAY
ADD    OVERPAY
STORE  GROSSPAY
```

Computing Software – Computer Languages

High-level languages:

Compiled Languages:

- Have English-like command syntax.
- Include languages such as Basic, Fortran, COBOL, Pascal, Ada, C, C++, and Java.
- Supported on many computer designs and *knowledge of the specific hardware is not as critical* in writing a program.
- System software called a **compiler** translates the program into machine language.

Interpreted languages: **MATLAB**

Interpreted and executed using an interpreter

Computing Software – Computer Languages

- **Interpreted languages** are easier to learn. An interpreter lets the programmer know immediately when and where problems exist in the code; compiled programs make the programmer wait until the program is complete.
- **Interpreters** therefore can be easier to use and produce more immediate results; however the source code of an interpreted language cannot run without the interpreter.
- **Compilers** produce better optimized code that generally run faster and compiled code is self sufficient and can be run on their intended platforms without the compiler present.

Computing Software – Computer Languages

Compilation Process:

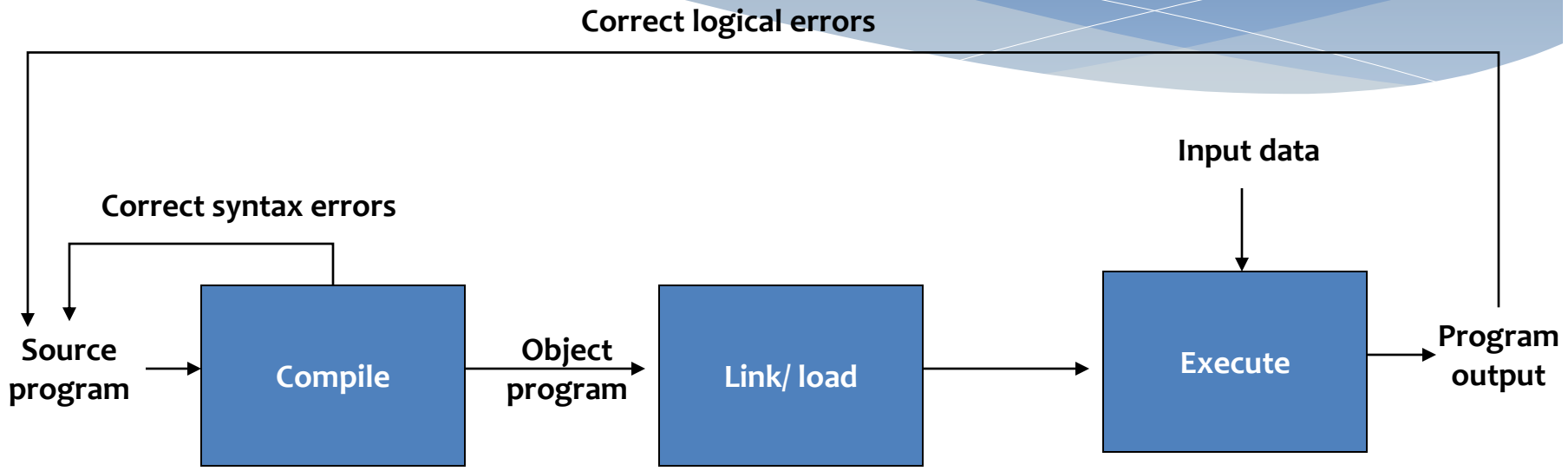
- Original program is called the **source program**
- Translated machine language program is called the **object program.**
- Errors detected by the compiler (called **compile errors** or **syntax errors**) must be corrected and compilation performed again.
- The process of compiling, correcting errors (or **debugging**) must often be repeated several times before the program compiles without compile errors.

Computing Software – Computer Languages

Execution Process:

- To prepare the object program for **execution**, system software is applied to **link** other machine language statements to the object program and then **load** the program into memory.
- The program is executed and new errors, called execution errors, run-time errors or **logic errors** (also called **bugs**) may be identified.
- These program errors are caused when the programmer errors in determining the correct steps to be taken in the problem solution, such as an attempt to *divide by zero*.
- These errors must be corrected in the source program, which must again be compiled and link-loaded for execution.

Computing Software – Computer Languages



Notice that even when a program executes without an error message, the results must be checked carefully to be sure that they are correct.

MATLAB: MATrix LABoratory

MATLAB
Mathematical Computation Tool

- **Matlab** is a computer program that combines computation and visualization power
- **Matlab** is both a computer programming language and a software environment for using that language effectively.
- The name **Matlab** stands for **MAT**rix **LAB**oratory, because the system was designed to make matrix computations particularly easy.
- The **Matlab** environment allows the user to manage variables, import and export data, perform calculations, generate plots, and develop and manage files for use with **Matlab**.

MATLAB
Mathematical Computation Tool

The Matlab environment is an **interactive** environment:

- **Single-line commands can be entered and executed.** This means that **you can type commands at the Matlab prompt and get answers immediately,** which is very useful for simple problems.
- **Matlab is an executable program,** developed in a high-level language, which *interprets* user commands.

MATLAB
Mathematical Computation Tool

- Portions of the Matlab program execute in response to the user input, results are displayed, and the program waits for additional user input.
- *Use of this environment doesn't require the compile-link/load-execution process* described above for high-level languages.

Engineering Problem Solving

* Engineering often involves applying *a consistent, structured* approach to the solving of problems.
* A general problem-solving approach and method can be defined, although variations will be required for specific problems.
* Problems must be approached *methodically*, applying an *algorithm*, or step-by-step procedure by which one arrives at a solution.

Engineering Problem Solving

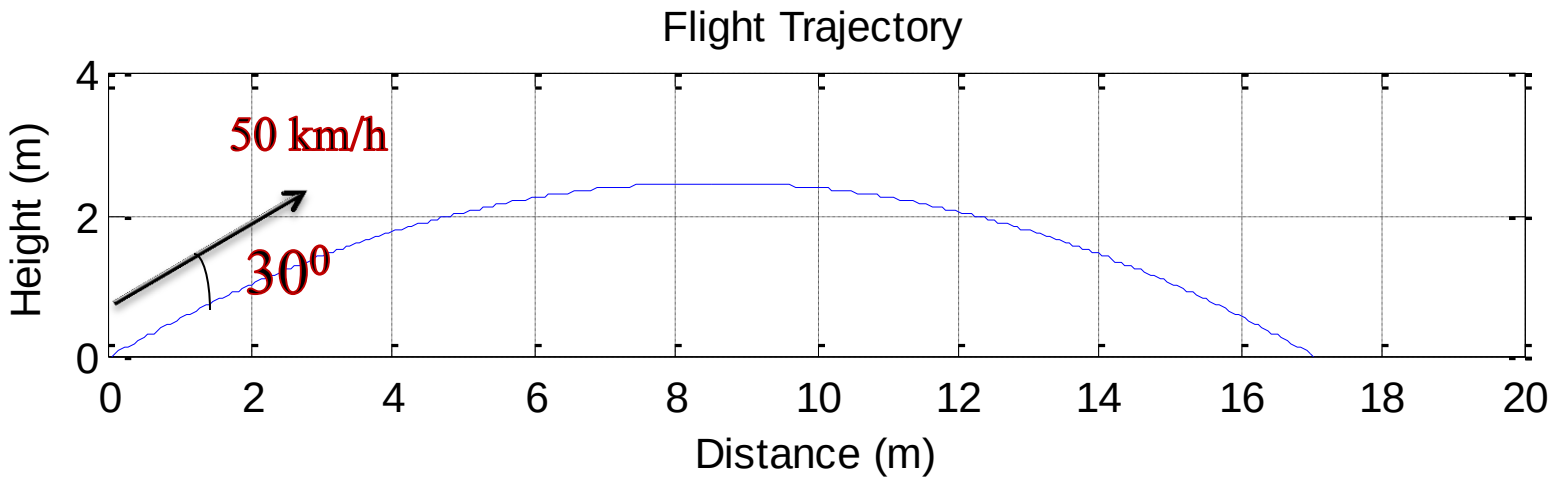
The **problem-solving process** for a computational problem can be outlined as follows:

1. Define the problem.
2. Create a mathematical model.
3. Develop a computational method for solving the problem.
4. Implement the computational method.
5. Test and assess the solution.

Problem Solving- Example

As an example of the problem solving process, consider the following problem:

A small object is launched into flight from the ground at a speed of 50 km/hour at 30 degrees above the horizontal over level ground. Determine the time of flight and the distance traveled when the ball returns to the ground.



Problem Solving Example - Definition

Assumptions:

Air resistance is negligible

Gravitational acceleration g = 9.8 m/s^2

unit conversions: km/h= 1000m/3600s

360 degrees = 2 π radians

Given:

Initial velocity: 50 km/hour

Angle: 30^0

Output:

the results to be produced are *the time of flight* (seconds) and *the distance traveled* (meters).

Problem Solving Example – Mathematical Model

Time: t (s), with $t = 0$ when the object is launched.
Initial velocity magnitude: $v = 50$ km/hour.
Initial angle: $\vartheta = 30^\circ$.
Horizontal position of ball: $x(t)$ (m).
Vertical position of ball: $y(t)$ (m).
Acceleration of gravity: $g = 9.8$ m/s^2, directed in negative y direction.

$$v_h = v\cos\theta$$

$$v_v = v\sin\theta$$

$$x(t) = vt\cos\theta$$

$$y(t) = vt\sin\theta - \frac{1}{2}gt^2$$

Problem Solving Example – Computational Model

Using the mathematical model, expressions for the desired results can be obtained. The object will hit the ground when its vertical position is zero:

$$y(t) = vt\sin\theta - \frac{1}{2}gt^2 = 0$$

Flight time:

$$t_g = \frac{2v\sin\theta}{g}$$

The horizontal position (distance of travel) at this time:

$$x(t_g) = vt_g\cos\theta$$

Problem Solving Example – Computational Implementation

* The equations defined in the computational method can be readily implemented using Matlab.
* *The commands used in the following solution will be discussed in detail later in the course*, but observe that the Matlab steps match closely to the solution steps from the computational method.

Problem Solving Example – Computational Implementation

```
% Flight trajectory computation
% Initial values
g = 9.8; % gravity, m/s^2
v = 50 * 1000/3600; % launch velocity, m/s
theta = 30 * pi/180;
disp('time of flight (s):') % label for time of flight
tg = 2 * v * sin(theta)/g % time to return to ground, s
disp('distance traveled (m):') % label for distance
xg = v * cos(theta) * tg
```

Problem Solving Example – Computational Implementation

```
% Compute and plot flight trajectory

t = linspace(0,tg,256);

x = v * cos(theta) * t;

y = v * sin(theta) * t - g/2 * t.^2;

plot(x,y), axis equal, axis([ 0 20 0 4 ]), grid, ...

xlabel('Distance (m)'), ylabel('Height (m)'), ...

title('Flight Trajectory')
```

Problem Solving Example – Comments:

- *Words following percent signs (%) are comments* to help in reading the Matlab statements.
- If a Matlab command assigns or computes a value, it will *display the value on the screen if the statement does not end with* ***a semicolon (;).***
- The ***three dots (...)*** at the end of a line mean that *the statement continues on the next line.*

Problem Solving Example – Comments:

- ***More than one statement*** can be entered on the same line if the statements are **separated by commas**.
- The statement **t = linspace(0,tg,256)**; *creates a* ***vector of length 256***.
- In addition to the required results, the flight trajectory has also been *computed and plotted*. This will be used below in assessing the solution.

Programming Style

Elements of good programming style include:

- *Use comments*, both at the beginning of a script, and also throughout the script to introduce different logical sections.
- *Describe each variable briefly* in a comment when it is initialized.
- *Separate sections of code* with blank lines.
- *Indent multiple line structures* to make them stand out as a logical unit.
- *Use spaces in expressions* to make them more readable (for example, on either side of operators and equal signs).