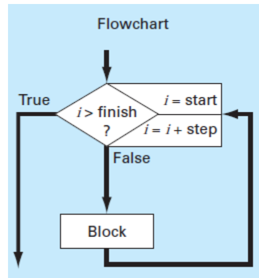
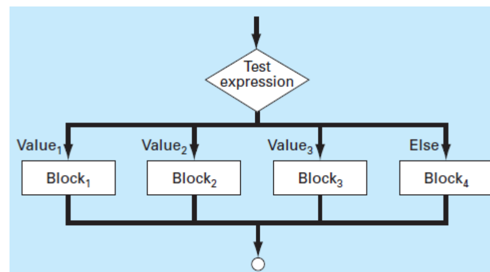
Lecture 02: Programming & Software

```
DOFOR i = start, finish, step
  Block
ENDDO
```





Bungee Jumper Problem

- Analytical solution: $\frac{dv}{dt} = g - \frac{c}{m}v$
- Numerical solution: $v_{i+1} = v_i + \frac{dv_i}{dt}\Delta t$
  - To obtain good accuracy, many small steps must be taken → extremely laborious and time-consuming to implement by hand.
- This lecture will help you figure out how to use computer as a tool to obtain numerical solutions.

Types of Software Users

1. First kind of users limit themselves to the capabilities found in the software's standard mode of operation.
2. Second kind of users are called "power user".
   - They learn to write programs of their own. For example, MATLAB m-files.

What is a Computer Program?

- A set of instructions that direct the computer to perform a certain task.
- We need the following programming capabilities:
  - Simple information representation (constants, variables, and type declarations).
  - Advanced information representation (data structure, arrays, and records).
  - Mathematical formulas (assignment, priority rules, and intrinsic functions).
  - Input/output.
  - Logical representation (sequence, selection, and repetition).
  - Modular programming (functions and subroutines).

Structured Programming

- In the early days of computer, programmers usually did not pay much attention to whether their programs were clear and easy to understand.
- Today, it is recognized that there are many benefits to writing organized, well-structured code.
- Aside from the obvious benefit of making software much easier to share, it also helps generate much more efficient program development.
- Well-structured algorithms are invariably easier to debug and test, resulting in programs that take a shorter time to develop, test, and update.

Structured Programming

- Structured programming is a set of rules that prescribe good style habits for the programmer.
- A key idea is that any numerical algorithm can be composed using the three fundamental control structures:
  - sequence,
  - selection, and
  - repetition.

Flowcharts

- A *flowchart is a visual or* graphical representation of an algorithm.
- The flowchart employs a series of blocks and arrows, each of which represents a particular operation or step in the algorithm.
- The arrows represent the sequence in which the operations are implemented.

Symbols used in Flowcharts

SYMBOL	NAME	FUNCTION
	Terminal	Represents the beginning or end of a program.
	Flowlines	Represents the flow of logic. The humps on the horizontal arrow indicate that it passes over and does not connect with the vertical flowlines.
	Process	Represents calculations or data manipulations.
	Input/output	Represents inputs or outputs of data and information.
	Decision	Represents a comparison, question, or decision that determines alternative paths to be followed.
	Junction	Represents the confluence of flowlines.
	Off-page connector	Represents a break that is continued on another page.
	Count-controlled loop	Used for loops which repeat a prespecified number of iterations.

Pseudocode

- An alternative approach to express an algorithm that bridges the gap between flowcharts and computer code is called *pseudocode.*
- This technique uses code-like statements in place of the graphical symbols of the flowchart.
- One advantage of pseudocode is that it is easier to develop a program with it than with a flowchart.
- The pseudocode is also easier to modify and share with others.

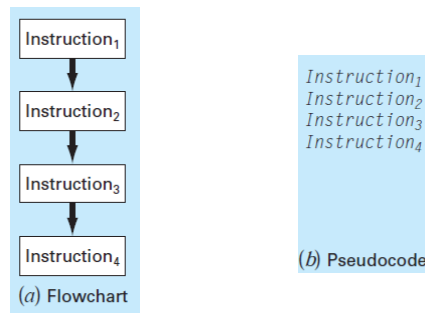
Style Conventions for Pseudocode

- Keywords such as IF, DO, INPUT, etc., are capitalized, whereas the conditions, processing steps, and tasks are in lowercase.
- Additionally, the processing steps are indented.

```
IF condition THEN
   True block
ENDIF
```

Logical Representation

- **Sequence:** The sequence structure expresses the trivial idea that unless you direct it otherwise, the computer code is to be implemented one instruction at a time.



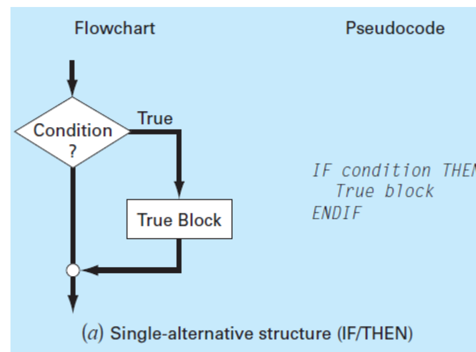
(a) Flowchart

Instruction1
Instruction2
Instruction3
Instruction4

(b) Pseudocode

Logical Representation

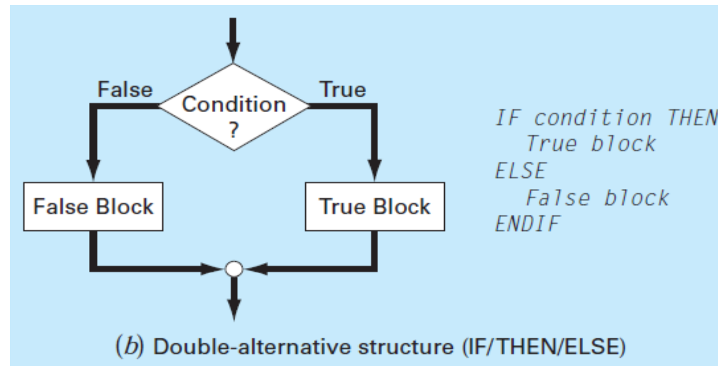
- **Selection:** In contrast to the step-by-step sequence structure, selection provides a means to split the program's flow into branches based on the outcome of a logical condition.



(a) Single-alternative structure (IF/THEN)

Logical Representation

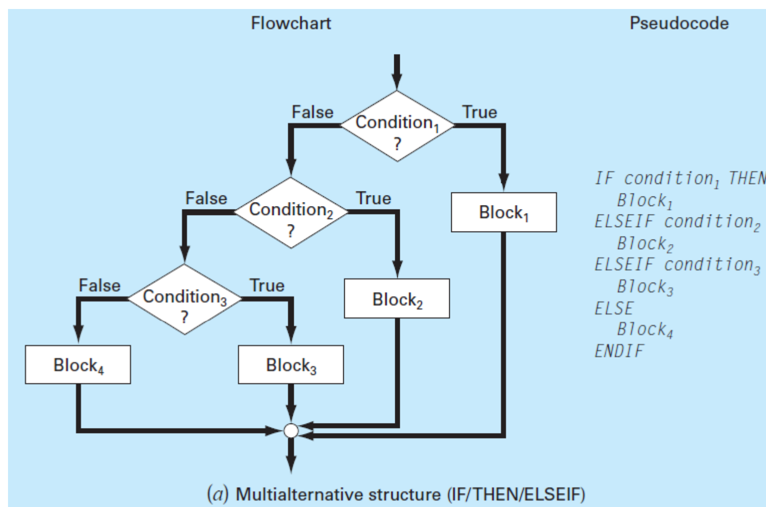
- **Selection:**



(b) Double-alternative structure (IF/THEN/ELSE)

Logical Representation

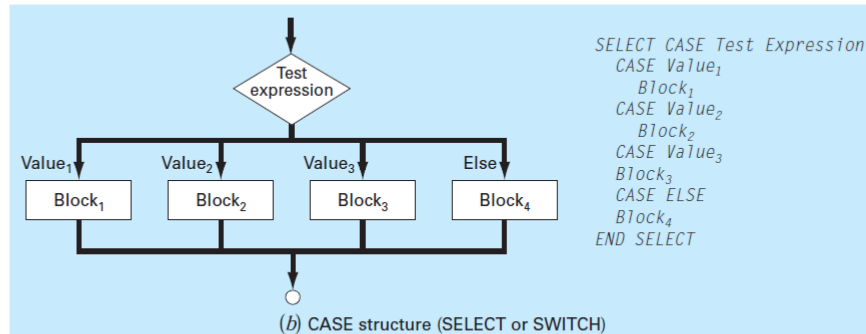
- **Selection:**



(a) Multialternative structure (IF/THEN/ELSEIF)

Logical Representation

- **Selection:**



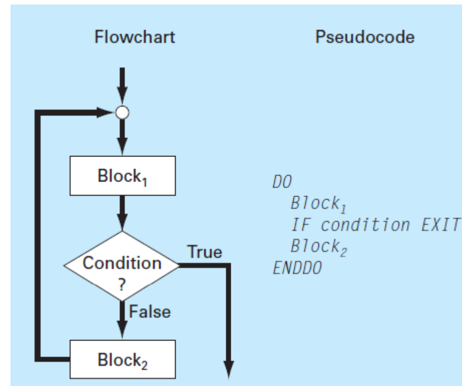
(*b*) CASE structure (SELECT or SWITCH)

Logical Representation

- **Repetition:** Repetition provides a means to implement instructions repeatedly. The resulting constructs, called loops, come in two "flavors" distinguished by how they are terminated.
- The first and most fundamental type is called a decision loop because it terminates based on the result of a logical condition.
- In contrast, a count-controlled or DOFOR loop performs a specified number of repetitions, or iterations.

Logical Representation

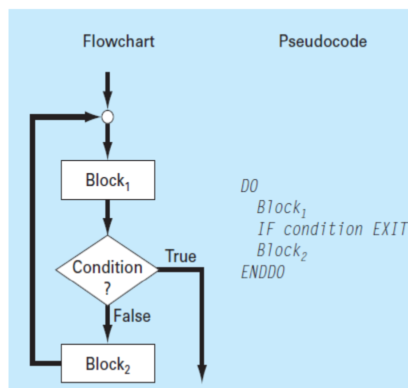
- **Repetition:** Decision loop



The most generic type of decision loop, the *DOEXIT construct, also called a break loop*

Logical Representation

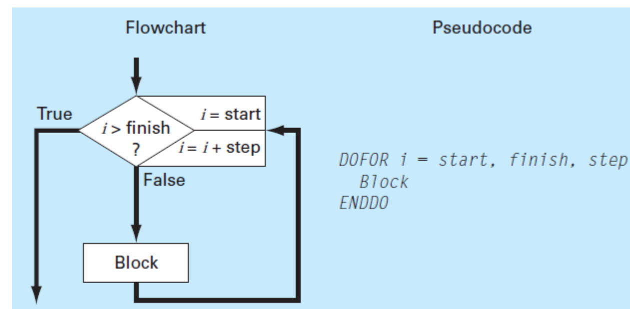
- **Repetition:** Decision loop



- If the first block is not included, the structure is sometimes called a *pretest loop*.
- Alternatively, if the second block is omitted, it is called a *posttest loop*.
- Because both blocks are included, it is sometimes called a *midtest loop*.

Logical Representation

- **Repetition:** Count-controlled or DOFOR construct



Example: Algorithm for Roots of a Quadratic

Problem Statement. The roots of a quadratic equation

$$ax^2 + bx + c = 0$$

can be determined with the quadratic formula,

$$\frac{x_1}{x_2} = \frac{-b \pm \sqrt{|b^2 - 4ac|}}{2a}$$

Develop an algorithm that does the following:

Step 1: Prompts the user for the coefficients, a, b, and c.
Step 2: Implements the quadratic formula, guarding against all eventualities (for example, avoiding division by zero and allowing for complex roots).
Step 3: Displays the solution, that is, the values for x.
Step 4: Allows the user the option to return to step 1 and repeat the process.

Example: Algorithm for Roots of a Quadratic

```
DO
  INPUT a, b, c
  r1 = (−b + SQRT(b^2 − 4ac))/(2a)
  r2 = (−b − SQRT(b^2 − 4ac))/(2a)
  DISPLAY r1, r2
  DISPLAY 'Try again? Answer yes or no'
  INPUT response
  IF response = 'no' EXIT
ENDDO
```


Example: Algorithm for Roots of a Quadratic

```
DO
  INPUT a, b, c
  r1 = 0: r2 = 0: i1 = 0: i2 = 0
  IF a = 0 THEN
    IF b ≠ 0 THEN
       r1 = −c/b
    ELSE
       DISPLAY "Trivial solution"
    ENDIF
  ELSE
    discr = b^2 − 4 * a * c
    IF discr ≥ 0 THEN
       r1 = (−b + Sqrt(discr))/(2 * a)
       r2 = (−b − Sqrt(discr))/(2 * a)
    ELSE
       r1 = −b/(2 * a)
       r2 = r1
       i1 = Sqrt(Abs(discr))/(2 * a)
       i2 = −i1
    ENDIF
  ENDIF
  DISPLAY r1, r2, i1, i2
  DISPLAY 'Try again? Answer yes or no'
  INPUT response
  IF response = 'no' EXIT
ENDDO
```

Example: Bungee Jumper Problem

$$v_{i+1} = v_i + \frac{dv_i}{dt}\Delta t$$

- Suppose that we started the computation at *t = 0 and* wanted to predict the velocity at *t* = *4 s* using a time step of *t* = 0.5 s → *n* = 4/0.5 = 8.

```
g = 9.8
INPUT cd, m
INPUT ti, vi, tf, dt
t = ti
v = vi
n = (tf – ti) / dt
DOFOR i = 1 TO n
  dvdt = g – (cd / m) * v
  v = v + dvdt * dt
  t = t + dt
ENDDO
DISPLAY v
```


Example: Bungee Jumper Problem

$$v_{i+1} = v_i + \frac{dv_i}{dt}\Delta t$$

- If the computation interval is not evenly divisible by the time step.

```
g = 9.8
INPUT cd, m
INPUT ti, vi, tf, dt
t = ti
v = vi
h = dt
DO
  IF t + dt > tf THEN
    h = tf – t
  ENDIF
  dvdt = g – (cd / m) * v
  v = v + dvdt * h
  t = t + h
  IF t ≥ tf EXIT
ENDDO
DISPLAY v
```

Modular Programming

- Computer programs can be divided into small subprograms, or modules, that can be developed and tested separately.
- This approach is called *modular programming*.
- The most important attribute of modules is that they be as independent and self-contained as possible.
- In addition, they are typically designed to perform a specific, well-defined function and have one entry and one exit point.
- Two types of procedures are commonly employed:
  - *Functions → usually returns a single result*
  - *Subroutines → usually returns several results*

Modular Programming

- Modular programming has a number of advantages:
  - The use of small, self-contained units makes the underlying logic easier to devise and to understand for both the developer and the user.
  - Development is facilitated because each module can be perfected in isolation.
  - Modular design also increases the ease with which a program can be debugged and tested because errors can be more easily isolated.
  - Program maintenance and modification are facilitated.
  - It allows you to maintain your own library of useful modules for later use in other programs.

Modular Programming

- Pseudocode for a function that solves a differential equation using Euler's method.

```
FUNCTION Euler(dt, ti, tf, yi)
t = ti
y = yi
h = dt
DO
  IF t + dt > tf THEN
    h = tf − t
  ENDIF
  dydt = dy(t, y)
  y = y + dydt * h
  t = t + h
  IF t ≥ tf EXIT
ENDDO
Euler = y
END
```


MATLAB Basics

- MATLAB was originally developed as a matrix laboratory.
- To this day, the major element of MATLAB is still the matrix.
- Programs can be written as so-called M-files that can be used to implement numerical calculations.
- Examples: M-files.

Fundamental Control Structures

(a) **Pseudocode**	(b) **MATLAB**
IF/THEN:	
IF condition THEN	`if b ~= 0`
True block	`r1 = -c / b;`
ENDIF	`end`
IF/THEN/ELSE:	
IF condition THEN	`if a < 0`
True block	`b = sqrt(abs(a));`
ELSE	`else`
False block	`b = sqrt(a);`
ENDIF	`end`

Fundamental Control Structures

(a) **Pseudocode**	(b) **MATLAB**
IF/THEN/ELSEIF:	
IF $condition_1$ *THEN*	`if class == 1`
$Block_1$	`x = x + 8;`
ELSEIF $condition_2$	`elseif class < 1`
$Block_2$	`x = x - 8;`
ELSEIF $condition_3$	`elseif class < 10`
$Block_3$	`x = x - 32;`
ELSE	`else`
$Block_4$	`x = x - 64;`
ENDIF	`end`

Fundamental Control Structures

(a) Pseudocode	(b) MATLAB

CASE:

```
SELECT CASE Test Expression          switch a + b
  CASE Value1                          case 1
    Block1                               x = -5;
  CASE Value2                          case 2
    Block2                               x = -5 - (a + b) / 10;
  CASE Value3                          case 3
    Block3                               x = (a + b) / 10;
  CASE ELSE                            otherwise
    Block4                               x = 5;
END SELECT                           end
```


Fundamental Control Structures

(a) Pseudocode	(b) MATLAB

DOEXIT:

```
DO                                   while (1)
  Block1                               i = i + 1;
  IF condition EXIT                    if i >= 10, break, end
  Block2                               j = i*x;
ENDDO                                end
```

COUNT-CONTROLLED LOOP:

```
DOFOR i = start, finish, step        for i = 1:2:10
  Block                                x = x + i;
ENDDO                                end
```

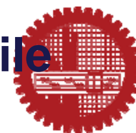
Example: Using a for Loop to Compute the Factorial

Problem Statement. Develop an M-file to compute the factorial.

$0! = 1$
$1! = 1$
$2! = 1 \times 2 = 2$
$3! = 1 \times 2 \times 3 = 6$
$4! = 1 \times 2 \times 3 \times 4 = 24$
$5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$
$\vdots$

```
function fout = factor(n)
% factor(n):
%   Computes the product of all the integers from 1 to n.
x = 1;
for i = 1:n
  x = x * i;
end
fout = x;
end
```


Example: Animation of Projectile Motion

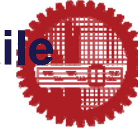


Problem Statement. In the absence of air resistance, the Cartesian coordinates of a projectile launched with an initial velocity (v_0) and angle (θ_0) can be computed with

$x = v_0 \cos(\theta_0)t$
$y = v_0 \sin(\theta_0)t - 0.5gt^2$

where $g = 9.81$ m/s^2. Develop a script to generate an animated plot of the projectile's trajectory given that $v_0 = 5$ m/s and $\theta_0 = 45°$.

Example: Animation of Projectile Motion



```
clc,clf,clear
g=9.81; theta0=45*pi/180; v0=5;
t(1)=0;x=0;y=0;
plot(x,y,'o','MarkerFaceColor','b','MarkerSize',8)
axis([0 3 0 0.8])
M(1)=getframe;
dt=1/128;
for j = 2:1000
  t(j)=t(j-1)+dt;
  x=v0*cos(theta0)*t(j);
  y=v0*sin(theta0)*t(j)-0.5*g*t(j)^2;
  plot(x,y,'o','MarkerFaceColor','b','MarkerSize',8)
  axis([0 3 0 0.8])
  M(j)=getframe;
  if y<=0, break, end
end
pause
movie(M,1)
```


Assignment-02

- Problems 2.1, 2.3, 2.4, 2.14, 2.22.