Chapter 3: Branching Statements and Program Design

Yunong Zhang (张雨浓)

Email: zhynong@mail.sysu.edu.cn

Top-down design techniques

- "Programming is easy. However, knowing what to do might be hard."

- 1/3: Planning what to do
  1/6: Writing the program
  1/2: Testing and debugging the program

Know more about this?

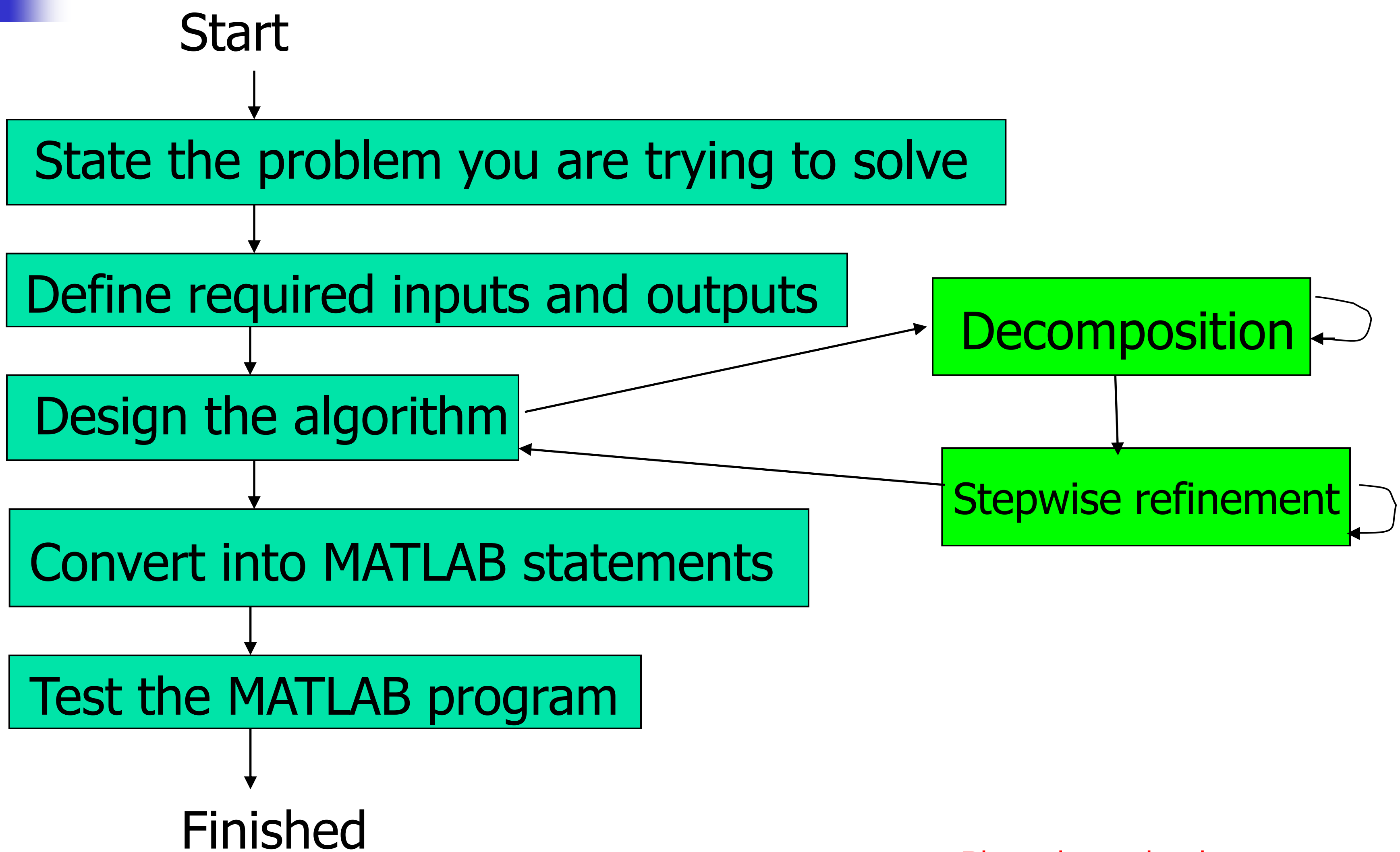
Top-down design techniques (Cont.)

- Top-down design is the process of starting with a large task and breaking it down into smaller, more easily understandable pieces, which perform a portion of the desired overall task.
- Each piece can be coded and tested independently.
- Combine the subtasks into a complete task after each subtask has been verified to work properly.

Top-down design techniques (Cont.)

1) Clearly state the problem that you are trying to solve
2) Define the inputs required by the program and the outputs to be produced by the program
3) Design the algorithm that you intend to implement so as to solve the problem
4) Turn the algorithm into MATLAB statements
5) Test the resulting MATLAB program

Top-down design techniques (Cont.)



Piecewise, pointwise

Relational and Logical Operators

- **Relational Operators:**

  Operators with two numerical or string operands that yield either true (1) or false (0); e.g.,

```
a1 op a2
```

where a1 and a2 are arithmetic expressions, variables, or strings, and op is one of the relational operators.

Relational Operators

Operator	Operation
==	Equal to
~=	Not equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

How is in JAVA/C?

Relational Operators (Cont.)

Operation	Result
3<4	1 (true)
3<=4	1
3==4	0 (false)
3>4	0
4<=4	1
‘A’<‘B’	1

Note: two strings must have the same lengths for comparison

abs(‘A’); setstr(67); abs('张雨浓')

Relational Operators (Cont.)

- Comparison of scalar-values in an array:

```
 >>a=[1 0
       -2 1 ];
```

c=[0 2;-2 -1];

```
>> b=0;
>>a>b
   [1 0; 0  1]
>>a>=c
   [1 0;
    1  1]
```

>> a.>=c

错误: 意外的 MATLAB 运算符

A caution about “==” and “~=”

a=0;

b=sin(pi);

Theoretically,

\>>a==b   1

But, in practice

\>>a==b
ans= 0

```
>> sin(pi)
ans = 1.2246e-016
>> eps
ans = 2.2204e-016
>> help eps
 EPS    Floating point relative accuracy.
    EPS returns the distance from 1.0 to the next largest
    floating point number. EPS is used as a default tolerance by
    PINV and RANK, as well as several other MATLAB functions.
    See also REALMAX, REALMIN.
 Overloaded methods: quantizer/eps.m, qfilt/eps.m, qfft/eps.m
```

The accuracy is considered:
abs(a-b)<1.0E-14

Logic Operators

Expression1:
a1 *op* a2
Expression2:
op a1

Operator	Operation
&	Logical AND
|	Logical OR
xor	Logical exclusive OR
~	Logical NOT

XOR is an important nonlinear-logic problem disturbing ANN around 1960-1970s.

Logic Operators (Cont.)

Inputs		and	or	xor	not
a1	a2	a1&a2	a1|a2	xor(a1,a2)	~a1
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0
		电路串	电路并		电路反相器

“时尚”电路：不同就对了！

Logic Operators (Cont.)

- Logic operation of scalar-values in an array:

```
>>a=[1 0; 0 1];
>>b=0;
>>a&b
 [0 0;0 0]
>>c=[1 1; 0 0];
>>a|c
[1 1;0 1]
```

```
>> xor(a,c)
ans =
    0    1
    0    1
```

Logic Operators (Cont.)

- Roughly, the order of operators' evaluation:

  1) All arithmetic operators are evaluated

  2) All relational operators (==,~=,>,>=,<,<=) are evaluated

  3) All "~", "&", "|" operators are evaluated

```
>> ~(3-7)>=1      ans = 0
>> ~((3-7)>=1)    ans = 1
```

```
>> ~2    ans = 0
>> ~3    ans = 0
>> ~4    ans = 0
>> ~-1   ans = 0
```

```
>> (0&~1)|~0   ans = 1
>> ~0&~1       ans = 0
>> ~(0&~1)     ans = 1
```

使用括号
简单有效

Additional Knowledge for Energetic Students

```
>> help ~
 Operators and special characters.

 Arithmetic operators.
   plus       - Plus                                +
   uplus      - Unary plus                          +
   minus      - Minus                               -
   uminus     - Unary minus                         -
   mtimes     - Matrix multiply                     *
   times      - Array multiply                     .*
   mpower     - Matrix power                        ^
   power      - Array power                        .^
   mldivide   - Backslash or left matrix divide     \
   mrdivide   - Slash or right matrix divide        /
   ldivide    - Left array divide                  .\
   rdivide    - Right array divide                 ./
   kron       - Kronecker tensor product          kron

 Relational operators.
   eq         - Equal                              ==
   ne         - Not equal                          ~=
   lt         - Less than                          <
   gt         - Greater than                       >
   le         - Less than or equal                 <=
   ge         - Greater than or equal              >=

 Logical operators.
   and        - Logical AND                        &
   or         - Logical OR                         |

   not        - Logical NOT                        ~

   xor        - Logical EXCLUSIVE OR
   any        - True if any element of vector is nonzero
   all        - True if all elements of vector are nonzero
```

… … … …

Logical functions

Function	Purpose
ischar(a)	Returns 1 if a is a character array and 0 otherwise
isinf(a)	Returns 1 if the value of a is infinite and 0 otherwise
isnumeric(a)	Returns 1 if the a is a numeric array and 0 otherwise

```
>>isinf(1/0)        1
>>isinf(1/10)       0

>>isinf([1/10,4,9])

ans = 0    0    0

>>isinf([1/10,4,inf])

ans = 0    0    1
```

```
>>isnumeric(2)       1
>>isnumeric('a')     0

>> isnumeric('a1o9j7e3')     ans = 0

>> isnumeric('a','1','o','9')

??? Error => Too many input arguments.

>> ischar('a1o9')            ans = 1

>> isnumeric('3573')         ans = 0

>> isnumeric(3573)           ans = 1
```

Testing Examples

a=2;

b=[1 -2;0 10];

c=[0 1;2 0];

d=1;

find the solutions of

1. ~(a>=b)
2. ~a>=b
3. a & b>=c
4. (a & b)>=c
5. b|c >a
6. (b|c)>a

Testing Examples (Cont.)

1)>>~(a>=b)

[0 0; 0 1]

2) cf. ~a>=b

ans = 0 1

1 0

3)>> a & b>=c

[1 0;0 1]

4) cf. (a & b)>=c

ans = 1 1

0 1

5)>> b|c >a

[1 1;0 1]

6) cf. (b|c)>a

[0 0;0 0]

使用括号，简单有效

Branches

- *if* construct
- *switch* construct
- *try/catch* construct

Branches -- if

>> help if

IF IF statement condition.

The general form of the IF statement is

```
IF expression
  statements
ELSEIF expression
  statements
ELSE
  statements
END
```

The statements are executed if the real part of the expression has all non-zero elements. The ELSE and ELSEIF parts are optional. Zero or more ELSEIF parts can be used as well as nested IF's.

The expression is usually of the form expr rop expr where

rop is ==, <, >, <=, >=, or ~=.

Example

```
if I == J
  A(I,J) = 2;
elseif abs(I-J) == 1
  A(I,J) = -1;
else
  A(I,J) = 0;
end
```

Branches -- switch

```
switch lower(METHOD)
    case {'linear','bilinear'}
     disp('Method is linear')
    case 'cubic'
     disp('Method is cubic')
    case 'nearest'
     disp('Method is nearest')
    otherwise
     disp('Unknown method')
  end
```

>> help switch

SWITCH Switch among several cases based on expression. The general form of the SWITCH statement is:

```
SWITCH switch_expr
  CASE case_expr,
    statement, ..., statement
  CASE {case_expr1, case_expr2, case_expr3,...}
    statement, ..., statement
 ...
  OTHERWISE,
    statement, ..., statement
END
```

The statements following the first CASE where the switch_expr matches the case_expr are executed. When the case expression is a cell array (as in the second case above), the case_expr matches if any of the elements of the cell array match the switch expression. If none of the case expressions match the switch expression then the OTHERWISE case is executed (if it exists). Only one CASE is executed and execution resumes with the statement after the END.

The switch_expr can be a scalar or a string. A scalar switch_expr matches a case_expr if switch_expr==case_expr. A string switch_expr matches a case_expr if strcmp(switch_expr,case_expr) returns 1 (true). Only the statements between the matching CASE and the next CASE, OTHERWISE, or END are executed. Unlike C, the SWITCH statement does not fall through (so BREAKs are unnecessary).

Example: To execute a certain block of code based on what the string, METHOD, is set to, method = 'Bilinear‘.

Branches -- try/catch

>> help try

TRY Begin TRY block.

The general form of a TRY statement is:

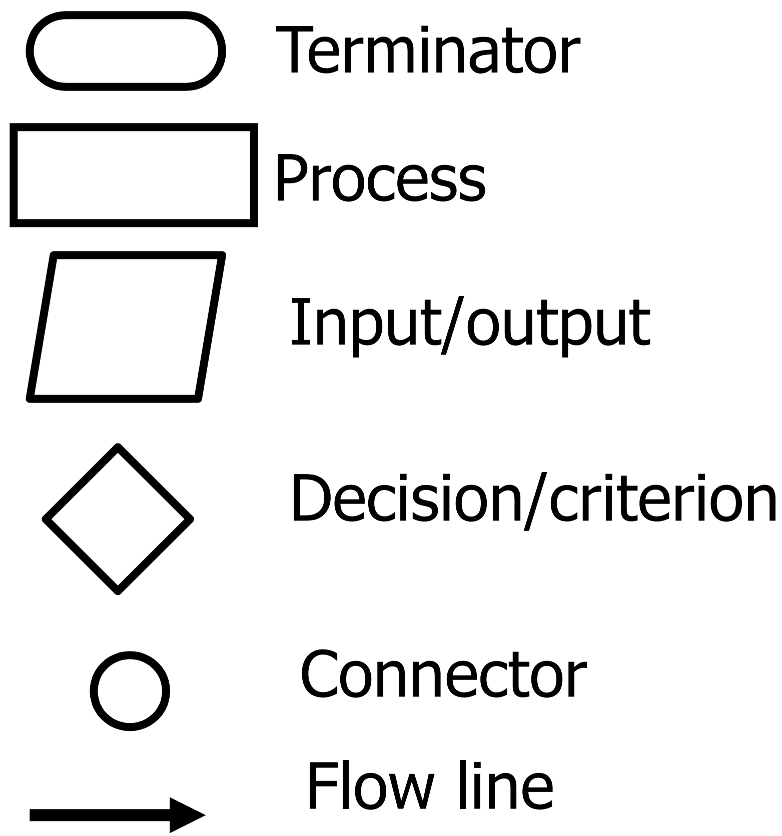
TRY, statement, ..., statement, CATCH, statement, ..., statement END

Normally, only the statements between the TRY and CATCH are executed. However, if an error occurs while executing any of the statements, the error is captured into LASTERR and the statements between the CATCH and END are executed. If an error occurs within the CATCH statements, execution will stop unless caught by another TRY...CATCH block. The error string produced by a failed TRY block can be obtained with LASTERR.

Help! Help!!

Flowcharts

- Graphical representation of algorithm

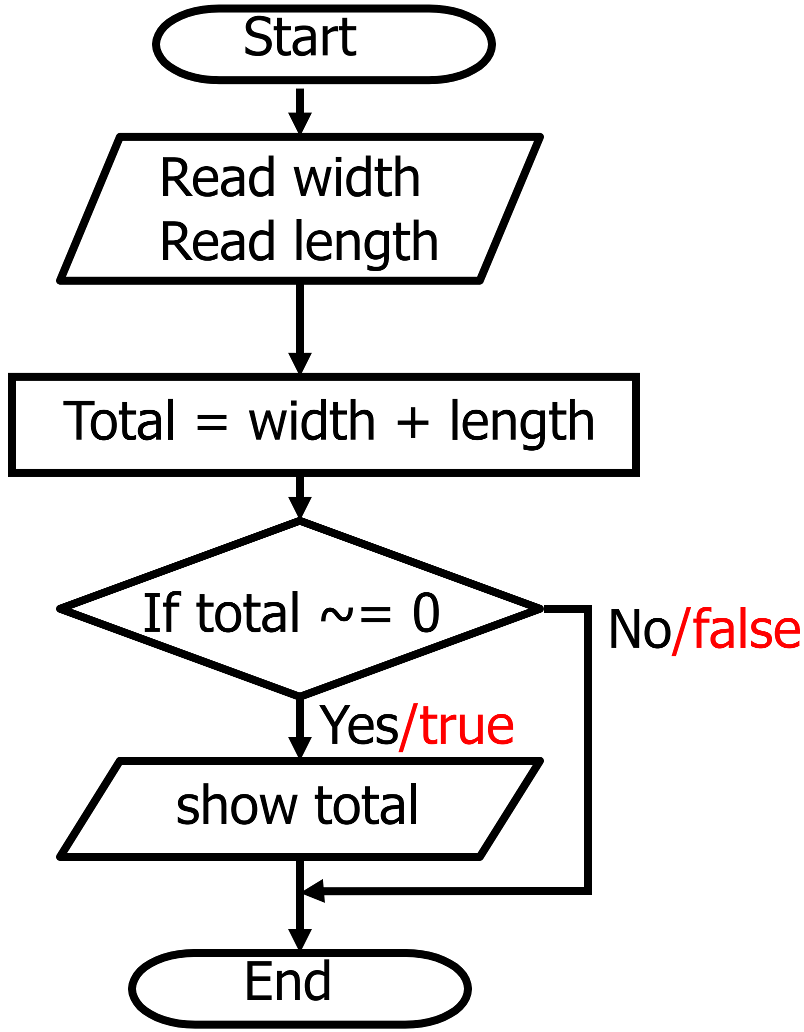


Flowchart example

Isn’t the total zero?

No, it is not zero.

Yes, it is.

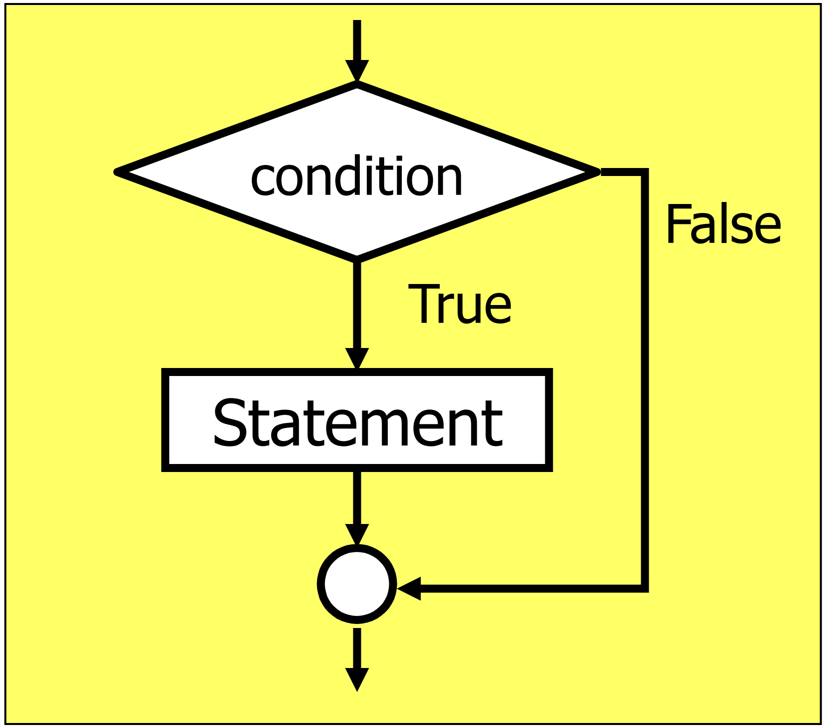


If Statement (In real life)

If I have free time,
(then) I will go to visit my friend**.** *(period)*

If Statement (In MATLAB)

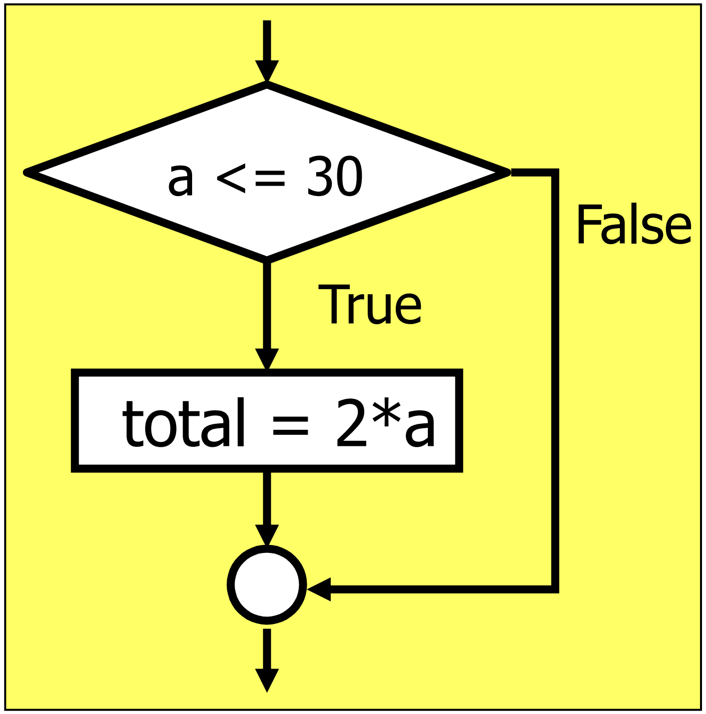
If logical expression
    statements
end



If Statement (Example)

```
If a <= 30
   total = 2*a
end
```

```
If a <= 30,
   total = 2*a;
end
```

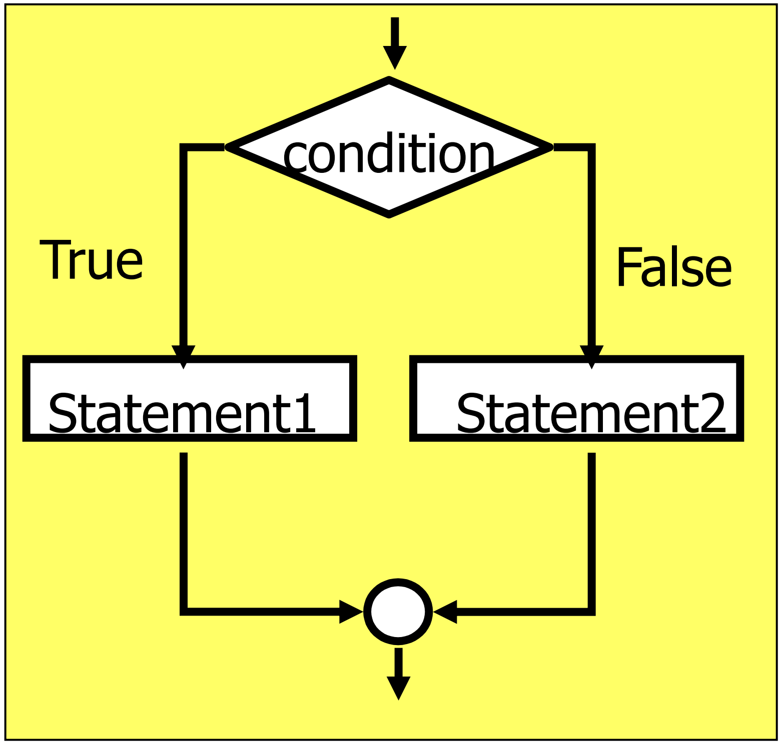


If-else Statement (In real life)

If I have free time,
  I will go to visit my friend;
otherwise (else),
  I will send an email to him**.** *(period)*

If-else Statement (In MATLAB)

If logical expression
    statement group 1
else
    statement group 2
end

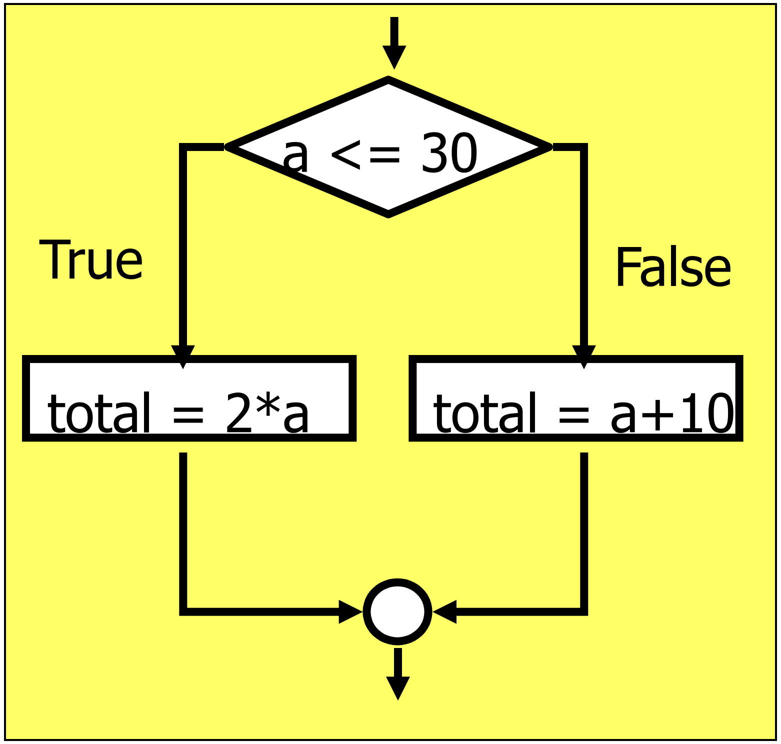


If-else Statement (Example)

```
If a <= 30
    total = 2*a
else
    total = a + 10
end
```

```
If a<=30
    total=2*a;
else
    total=a+10;
end
```

My preferred writing style

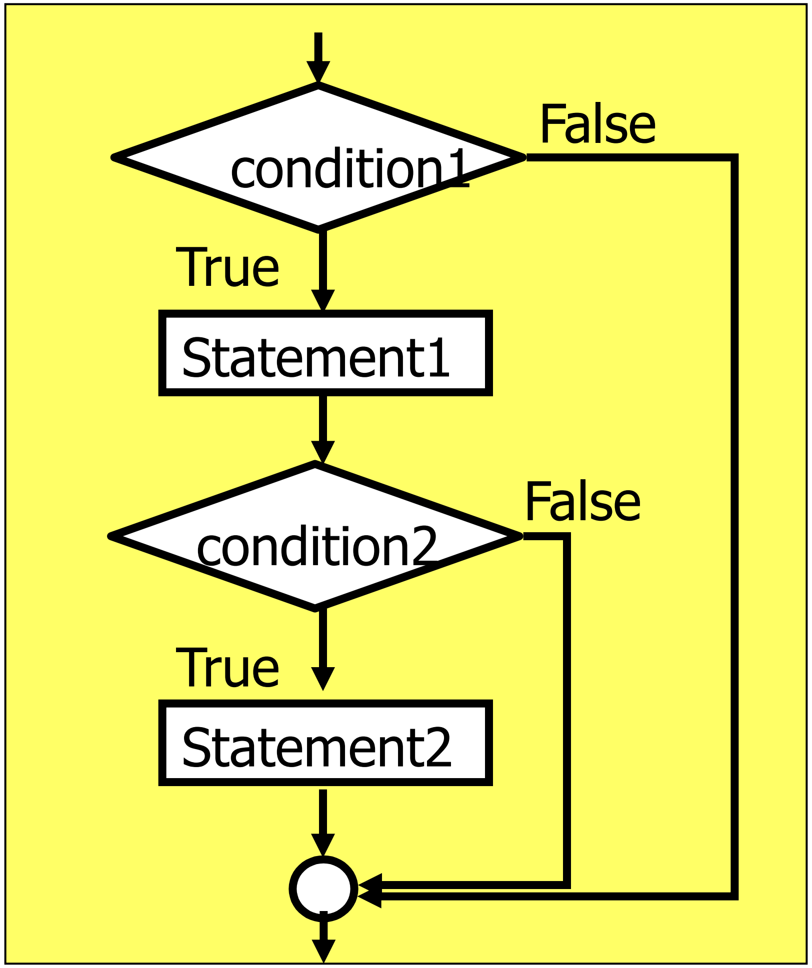


Nested Logic - I (In real life)

If I have free time,
I will go to visit my friend.
(But/and) if she is not there,
I will leave a message**.** *(period)*

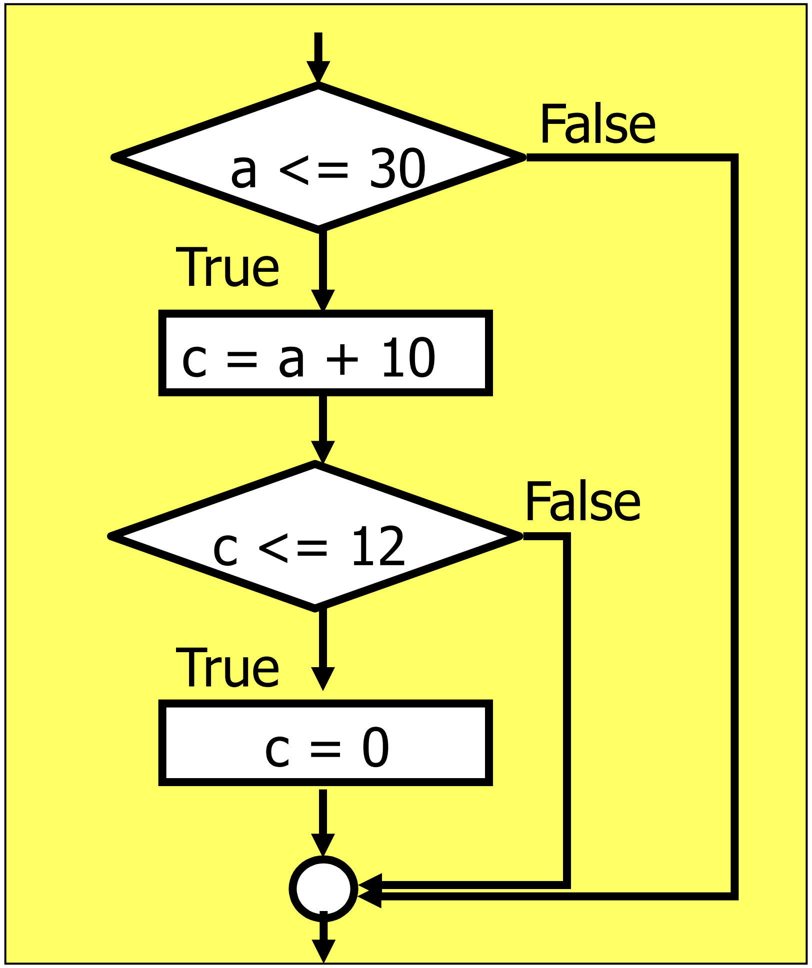
Nested Logic – I (In MATLAB)

If logical expression 1
    statement group 1
    If logical expression 2
        Statement group 2
    end
end



Nested Logic – I (Example)

```
If a <= 30
    c = a + 10
    If c <= 12
        c = 0
    end
end
```



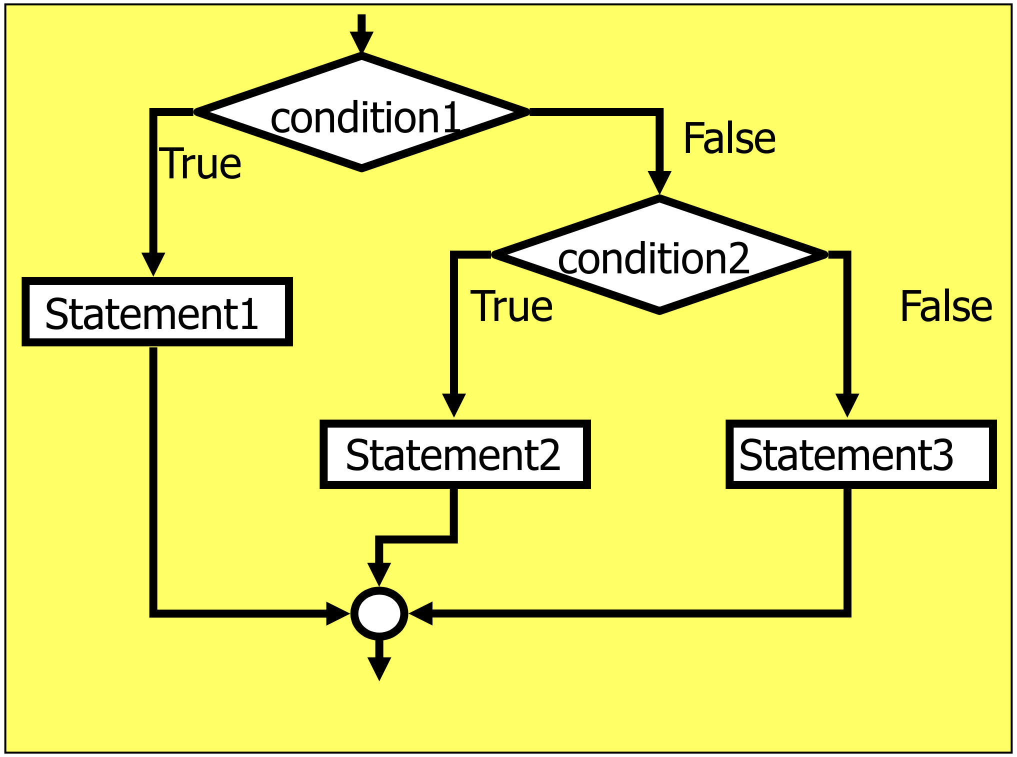
Nested Logic – II (In real life)

If I have free time,
  I will go to visit my friend
elseif I can access computer
  I will send her an email
else
  I will send her a note**.** *(period)*

The above is just an imaginary usage in our real-life.
  If…
  Otherwise, if…
  Otherwise, if the above conditions do not hold, …

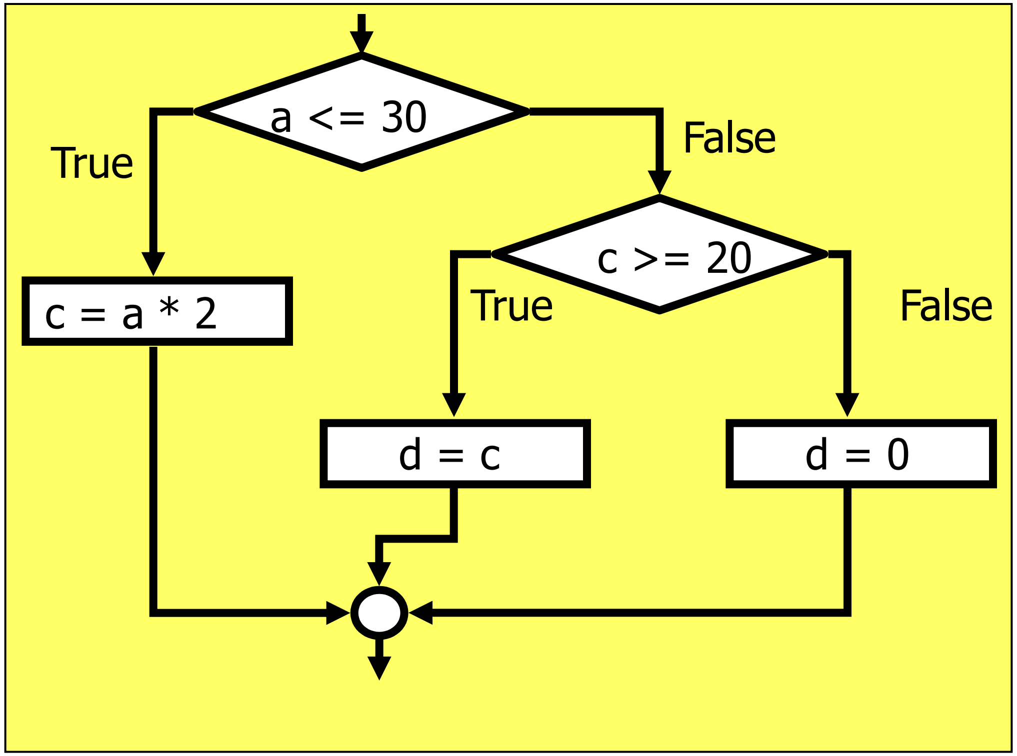
Nested Logic – II (In MATLAB)

If logical expression 1
      statement group 1
elseif logical expression 2
      statement group 2
else
      statement group 3
end



Nested Logic – II (Example)

```
If a <= 30
      c = a * 2
elseif c >= 20
      d = c
else
      d = 0
end
```



Notes: *if* constructs

- Equivalence
  multiple *elseif* clauses
  nested *else-if* constructs

- Assigning letter grades

grade	>95	A
95=>grade	>86	B
86=>grade	>76	C
76=>grade	>66	D
66=>grade	>0	F

A+	4.3	Excellent
A	4.0	
A-	3.7	
B+	3.3	Good
B	3.0	
B-	2.7	
C+	2.3	Adequate
C	2.0	
C-	1.7	
D	1.0	Marginal
F	0.0	Failure

Using multiple elseif clauses

- Using elseif clauses

```
if grade >95.0
   disp('The grade is A.');
elseif grade>86.0
   disp('The grade is B.');
elseif grade>76.0
   disp('The grade is C.');
elseif grade >66.0
   disp('The grade is D.');
else
   disp('The grade is F.');
end
```

Using nested else-if constructs

```
if grade >95.0
   disp('The grade is A.');
 else
   if grade >86.0
     disp('The grade is B.');
   else
     if grade >76.0
       disp('The grade is C.');
     else
       if grade >66.0
         disp('The grade is D.');
       else
         disp('The grade is F.');
       end
     end
   end
 end
```

Examples using *if* constructs

- Design and write a program to solve for the roots of the following equation:

  $ax^2+bx+c=0$

- **Solution**:

Step 1. Clearly state the problem:

To get the roots of a quadratic equation,

Note: it may have distinct real roots, repeated real roots, or complex roots.

Examples using *if* constructs (cont.)

Step 2. Define the inputs and outputs

For a quadratic equation

$ax^2+bx+c=0$

its inputs are: a, b, and c;

its outputs are: roots x

Examples using *if* constructs (cont.)

Step 3. Design the algorithm

Break the task down into three major sections

a. Read the input data;

b. Compute the roots;

c. Write out the roots

Examples using *if* constructs (cont.)

Step 4. Turn the algorithm into Matlab statements

% a	coefficient of x^2 term of equation
% b	coefficient of x term of equation
% c	constant of equation
% discriminant	discriminant of the equation
% imag_part	image part of complex roots
% real_part	real part of complex roots
% x1	the first solution of equation
% x2	the second solution of equation

How to translate the word 'discriminant'?

Examples using *if* constructs (cont.)

```
a=input('Enter the coefficient a:');
b=input('Enter the coefficient b:');
c=input('Enter the coefficient c:');

discriminant=b^2-4*a*c;
```

Examples using *if* constructs (cont.)

```
if discriminant>0
   x1=(-b+sqrt(discriminant))/(2*a);
   x2=(-b-sqrt(discriminant))/(2*a);
   fprintf('x1=%f\n',x1);
   fprintf('x2=%f\n',x2);
elseif discriminant==0
    x1=(-b)/(2*a);
    fprintf('x1=x2=%f\n',x1);
```

Examples using *if* constructs (cont.)

```
else
   real_part=(-b)/(2*a);
   imag_part=sqrt(abs(discriminant))/(2*a);
   fprintf('x1=%f +i %f\n', real_part,imag_part);
   fprintf('x2=%f -i %f\n', real_part,imag_part);
end
```

Examples using *if* constructs (cont.)

Step 5. Test the program.

[a b c]=[1 5 6], x1=-2 x2=-3

[a b c]=[1 4 4], x1=x2=-2

[a b c]=[1 2 5], x1=-1+2i, x2=-1-2i

>>Enter the coefficient a:1

>>Enter the coefficient b:5

>>Enter the coefficient c:6

x1=-2.0000, x2=-3.0000

Sincere Thanks!

- Using this group of PPTs, please read

- [1] Yunong Zhang, Weimu Ma, Xiao-Dong Li, Hong-Zhou Tan, Ke Chen, MATLAB Simulink modeling and simulation of LVI-based primal-dual neural network for solving linear and quadratic programs, Neurocomputing 72 (2009) 1679-1687

- [2] Yunong Zhang, Chenfu Yi, Weimu Ma, Simulation and verification of Zhang neural network for online time-varying matrix inversion, Simulation Modelling Practice and Theory 17 (2009) 1603-1617