Computer Science

MATLAB

Essential MATLAB for Scientists

Chap 3: Fundamentals (continued 2)

7. Repeating with for

- Avoid *for* loops by vectorising

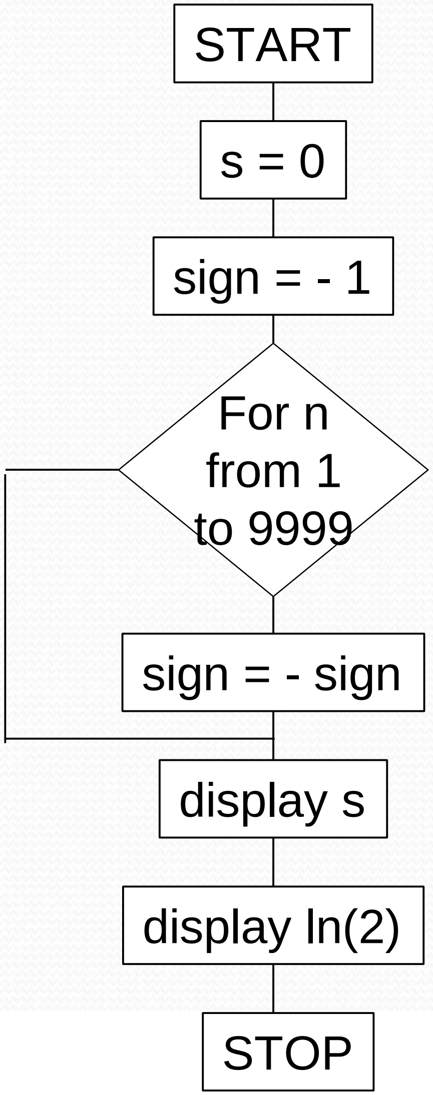
Series with alternating signs are a little more challenging. The following series sums to ln(2).

$$1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\frac{1}{5}-\cdots$$

- Think of an algorithm that finds the sum of the first 9999 terms of that sequence/series using a *for* loop then vectorize the calculation.

7. Repeating with for

- The series is $1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \cdots$



START

s = 0

sign = - 1

For n from 1 to 9999

sign = - sign

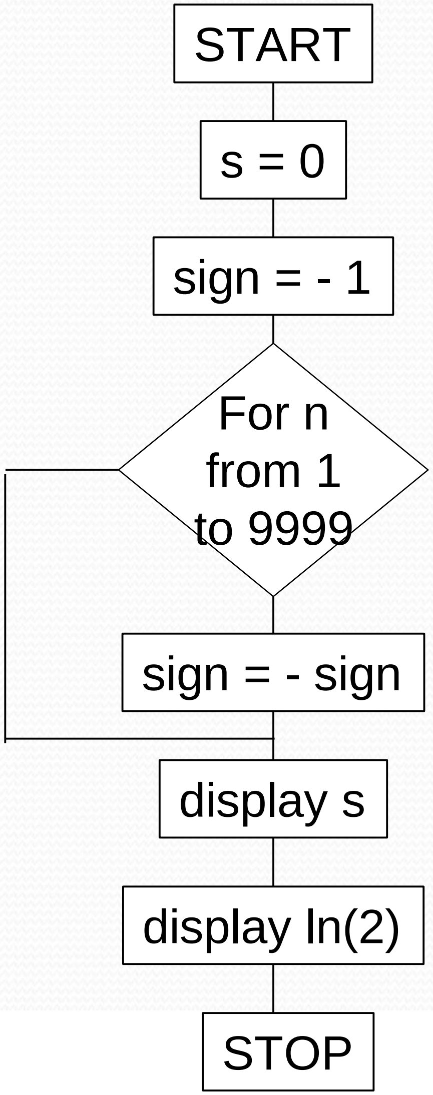
display s

display ln(2)

STOP

7. Repeating with for

- The series is $1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\frac{1}{5}-\cdots$



START

s = 0

sign = - 1

For n from 1 to 9999

sign = - sign

display s

display ln(2)

STOP

Program: Matlab code

- sign = -1;
- s = 0;
- for n = 1 : 9999
  - sign = - sign; or sign = -(-1)^n
  - s = s + sign*1/n;
- end
- disp(s);
- disp (log(2));

7. Repeating with for

- A common mistake: for less loops
  - A common mistake is to omit the word *for* from a *for* loop. Instead of reporting an error, MATLAB creates a vector, and executes the statements in the 'loop' only one. For example, run the program for square-rooting with Newton's method, leaving out the word *for*, i.e.
    - a = 2;
    - x = a/2;
    - i= 1:6
      - x = (x + a / x) / 2;
      - disp(x);
    - end
  - i is now a vector. Therefore, the loop is inexistent.

7. Repeating with for

- Exercises

Write algorithms and MATLAB programs to find the following sums

a) with for loops, and

b) by vectorization.

Time both versions in each case.

a) $1^2 + 2^2 + 3^2 + \ldots + 1000^2$

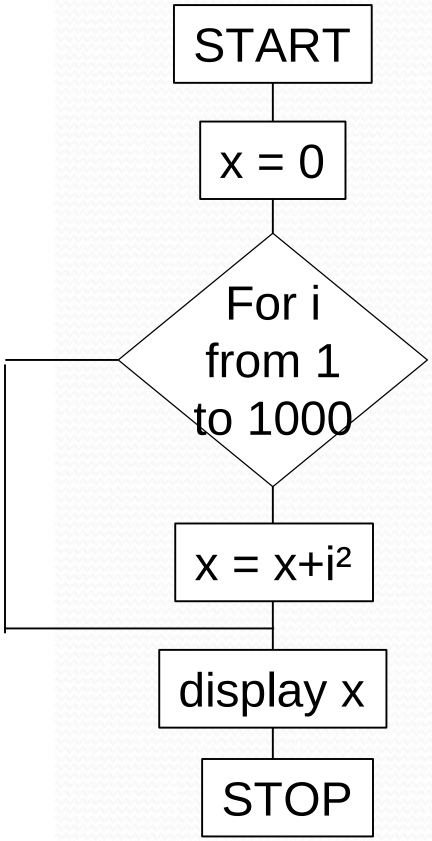
b) $-1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots - \frac{1}{1003}$

c) $\frac{1}{1^2.3^2} + \frac{1}{3^2.5^2} + \frac{1}{5^2.7^2} + \cdots + \frac{1}{999^2.1001^2}$ $(500\ terms)$

7. Repeating with for

a) $1^2 + 2^2 + 3^2 + \ldots + 1000^2$

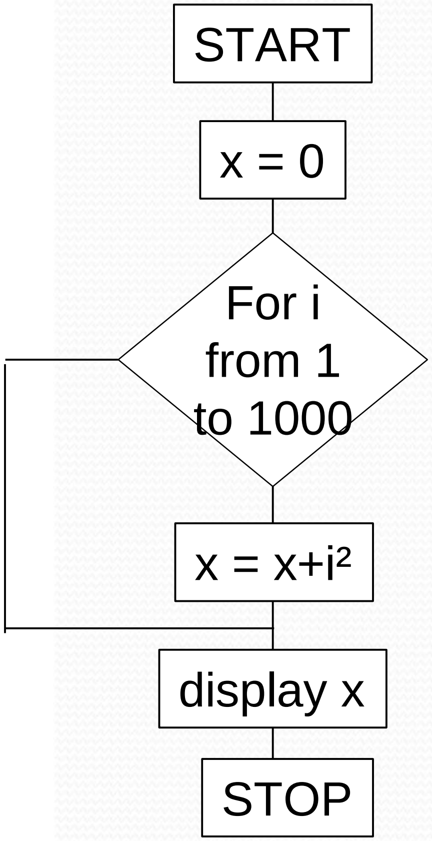
Algorithm with for loop 1st way



7. Repeating with for

a) $1^2 + 2^2 + 3^2 + ... + 1000^2$

Algorithm with for loop 1st way



START

x = 0

For i from 1 to 1000

x = x+i^2

display x

STOP

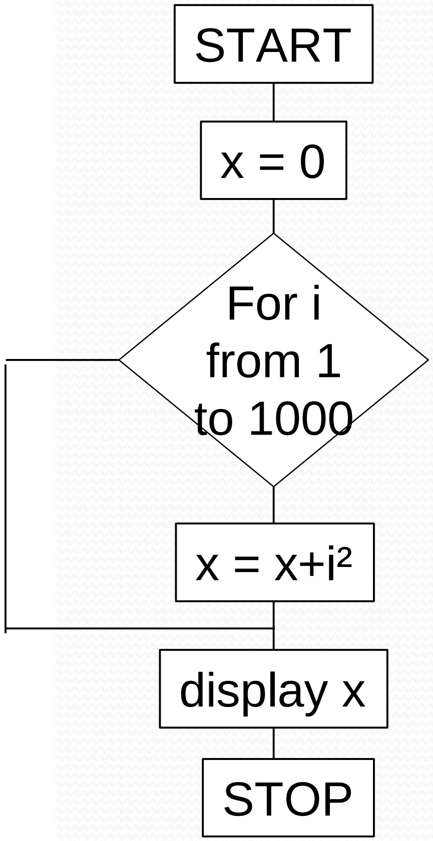
Program Matlab code

```
%With for loop, 1st way
x=0;
for i=1:1000
x=x+i^2;
end
```

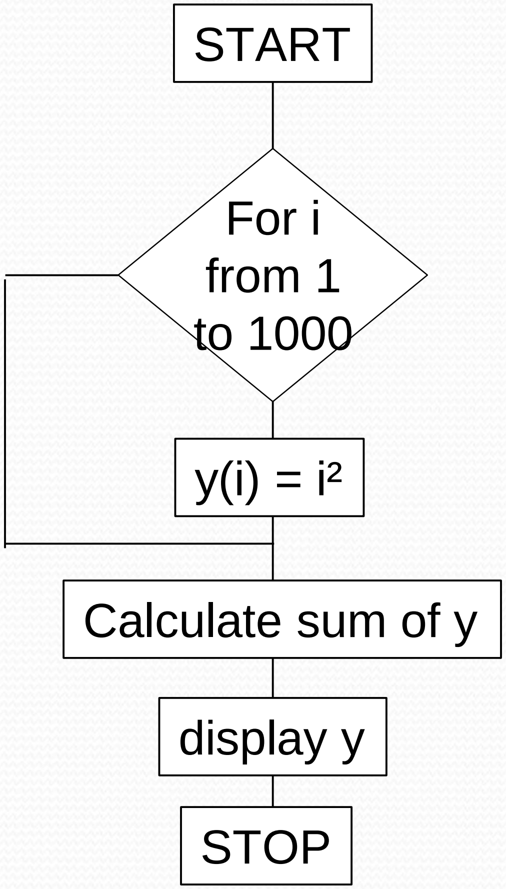
7. Repeating with for

a) $1^2 + 2^2 + 3^2 + \ldots + 1000^2$

Algorithm with for loop 1st way



Algorithm with for loop 2nd way



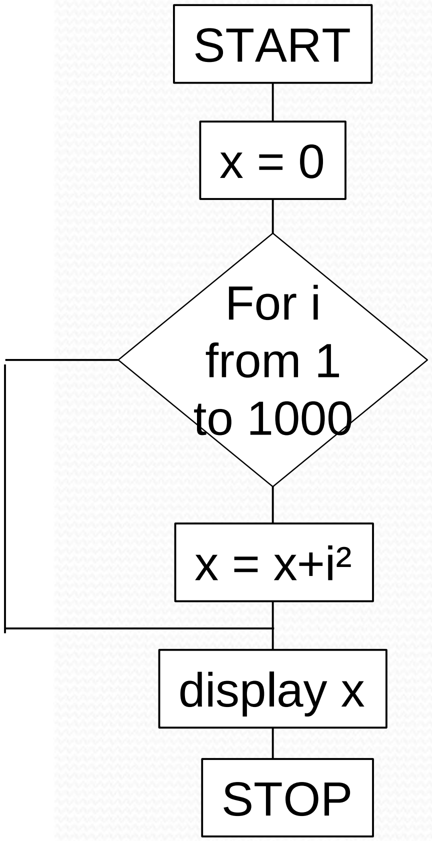
Program Matlab code

```
%With for loop, 1st way
x=0;
for i=1:1000
x=x+i^2;
end
```

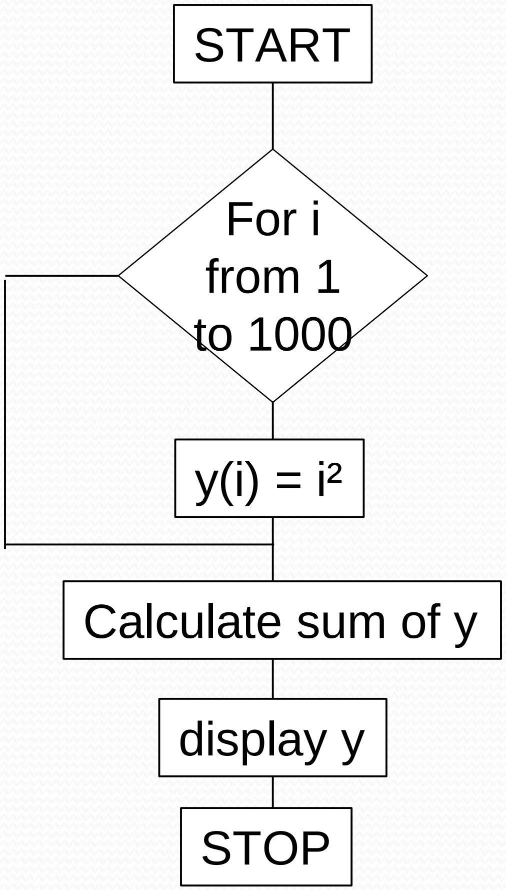
7. Repeating with for

a) $1^2 + 2^2 + 3^2 + \ldots + 1000^2$

Algorithm with for loop 1st way



Algorithm with for loop 2nd way



Program Matlab code

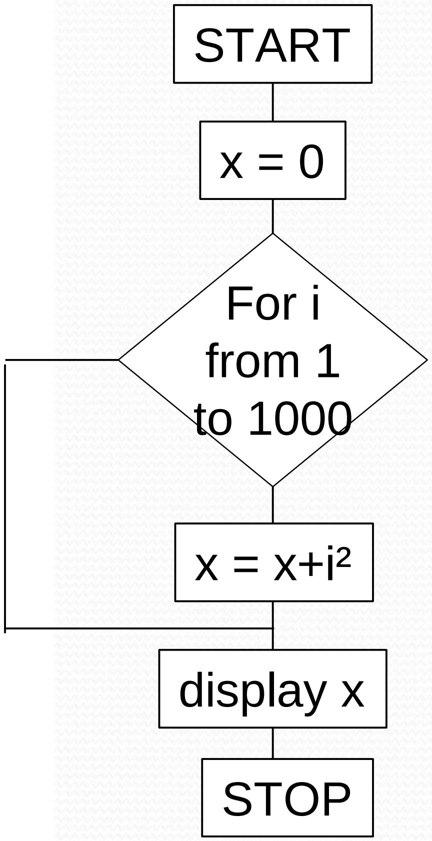
```
%With for loop, 1st way
x=0;
for i=1:1000
x=x+i^2;
end

%With for loop, 2nd way
for i=1:1000
y(i)=i^2;
end
sum(y)
```

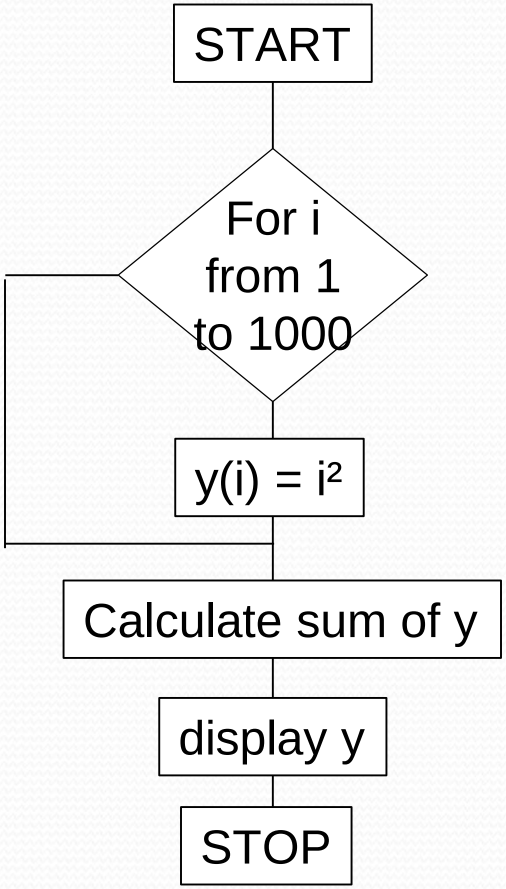
7. Repeating with for

a) $1^2 + 2^2 + 3^2 + ... + 1000^2$

Algorithm with for loop 1st way



Algorithm with for loop 2nd way



Program Matlab code

```
%With for loop, 1st way
x=0;
for i=1:1000
x=x+i^2;
end

%With for loop, 2nd way
for i=1:1000
y(i)=i^2;
end
sum(y)

%With vectorization
z=[1:1:1000];
sum(z.^2)
% x,sum(y) and sum(z.^2) have the same resu
```

8. Decisions

- Now enter the following:
  - 2 > 0
- Now enter the following:
  - -1 > 0
- MATLAB gives a value of 1 to a logical expression which is *true*, and 0 to one which is *false*.

8. Decisions

Exercises: The following statements all assign logical expressions to the variable *x*. See if you can correctly determine the value x in each case before checking your answer with MATLAB.

- x = 3 > 2
- x = 2 > 3
- x = -4 <= -3
- x = 1 < 1

- x = 2 ~= 2
- x = 3 == 3
- x = 0 < 1.5 + 1

8. Decisions

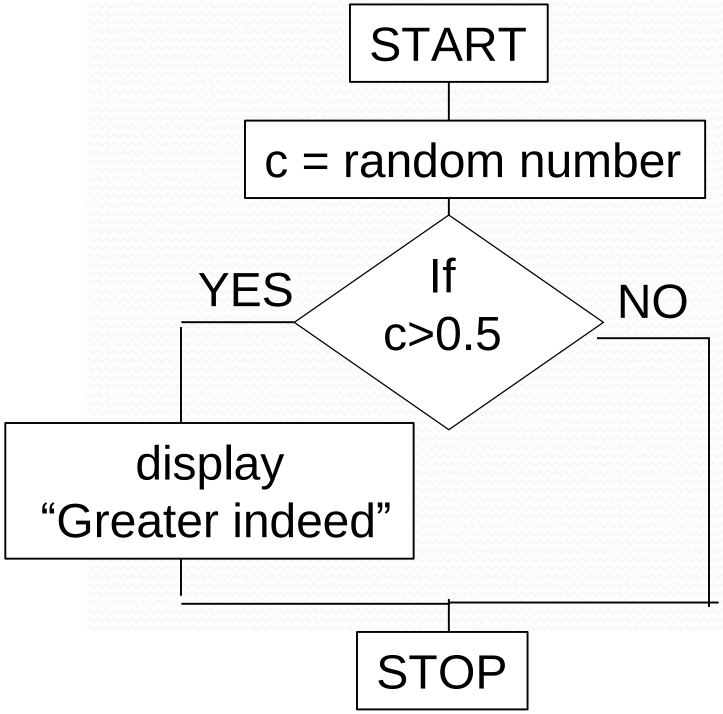
- The « if » structure

if *condition* is true, *statement* is executed. But if *condition* is false, nothing happens.

Example:

Would like to know if c (a random number) is greater than 0.5, display « greater indeed ».

Algorithm



8. Decisions

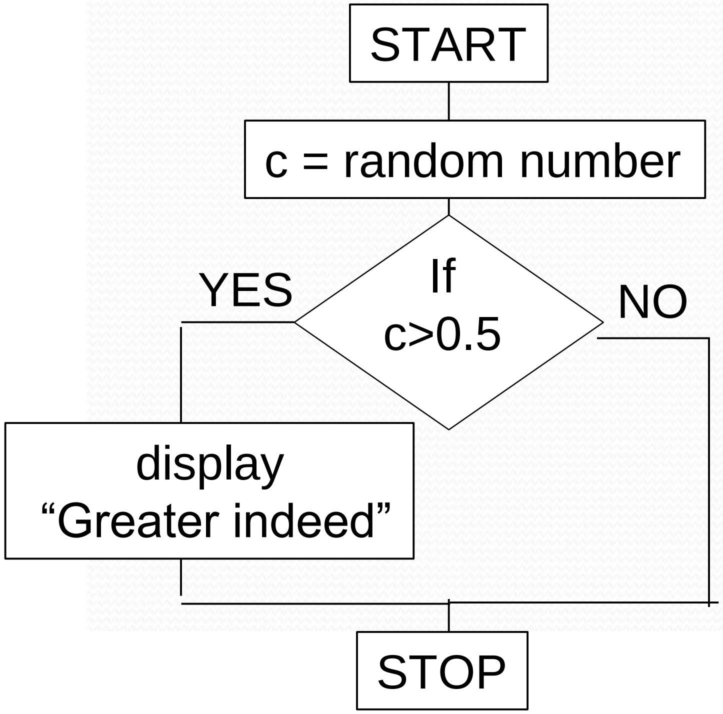
- The « if » structure

if *condition* is true, *statement* is executed. But if *condition* is false, nothing happens.

Example:

Would like to know if c (a random number) is greater than 0.5, display « greater indeed ».

Algorithm



START

c = random number

If c>0.5

YES

NO

display “Greater indeed”

STOP

Program Matlab code

```
if c > 0.5
    disp( ‘greater indeed’ )
end
```

8. Decisions

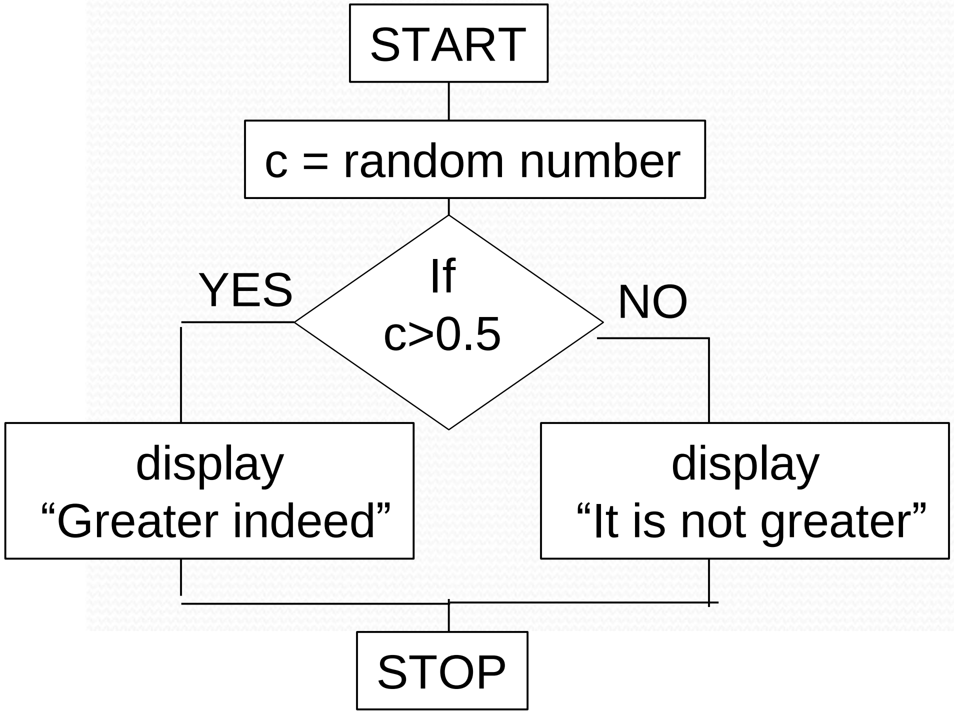
- The « if » structure

if *condition* is true, *statement* is executed. But if *condition* is false, nothing happens.

Example:

Would like to know if c (a random number) is greater than 0.5, display « greater indeed », else display « It is not greater ».

Algorithm



8. Decisions

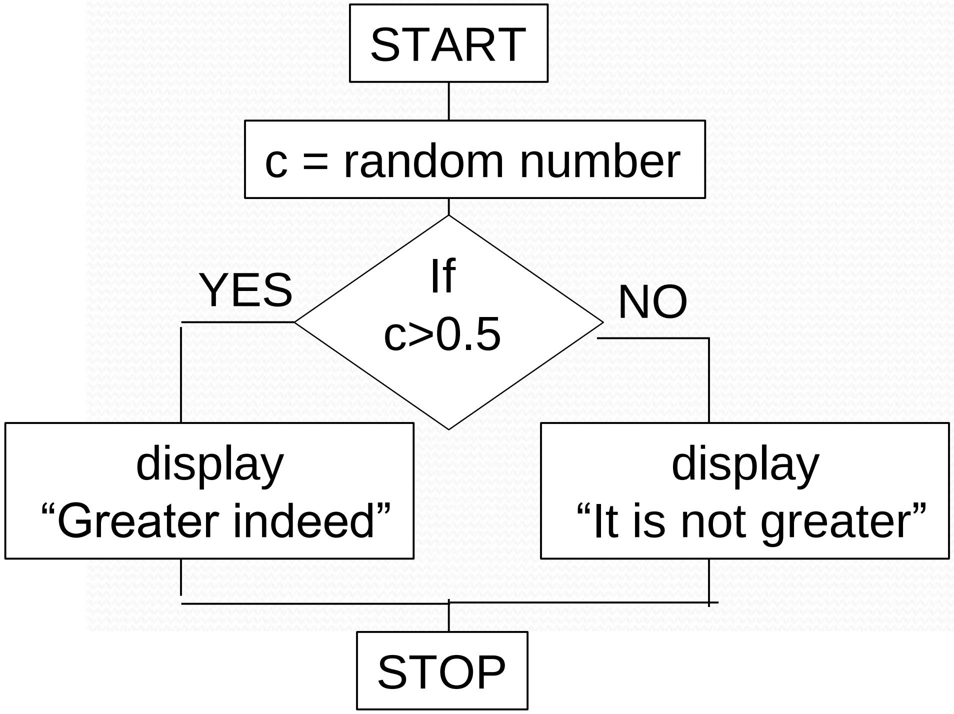
- The « if » structure

if *condition* is true, *statement* is executed. But if *condition* is false, nothing happens.

Example:

Would like to know if c (a random number) is greater than 0.5, display « greater indeed », else display « It is not greater ».

Algorithm



Program Matlab code

```
if c > 0.5
     disp( 'greater indeed' )
else
     disp( 'it is not greater' )
end
```

8. Decisions

- In the example of the previous slide, MATLAB has to make a decision depending on the value of *c*. The *if* construct, which is fundamental to all computing languages, is the basis of such decision-making.
- The simplest form of *if* in a single line is
  - if condition *(then)* statement, end

8. Decisions

- *condition* is usually a *logical expression*, i.e. an expression containing *relational operator*, and which is either *true* or *false*. The relational operators are shown in the following table.

Relational operator	Meaning
<	less than
<=	less than or equal
==	equal
~=	not equal
>	greater than
>=	greater than or equal

8. Decisions

- *condition* may be a vector or matrix, in which case it is true only if *all* its elements are non-zero. A single zero element in a vector or matrix renders it false.
- MATLAB allows you to use an arithmetic expression for *condition*. If the expression is false, then it evaluates to 0. Otherwise, it evaluates to 1.

8. Decisions

Exercise:

```
x = 2;
if x < 0
disp( ‘neg’ )
else
disp( ‘non-neg’ )
end
```

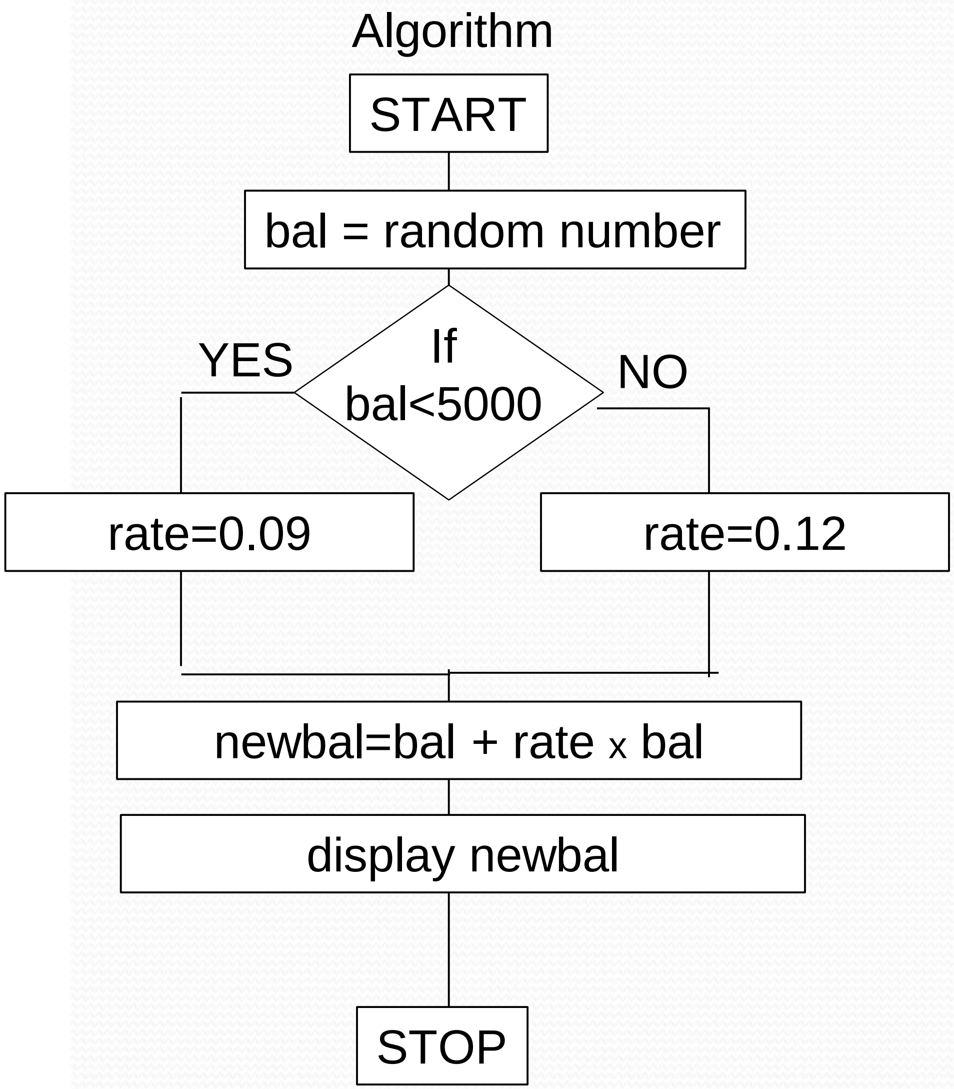
- Now change the value of x to -1 and execute the *if* again.

8. Decisions

Exercise:

- Let us consider the following example. Banks offer differential interest rates. Suppose the rate is 9% if the amount in your savings account is less than $5000, but 12% otherwise. The Random Bank goes one step further and gives you a random amount in your account to start with! Type and run the following program a few times.

8. Decisions



Algorithm

START

bal = random number

If bal<5000

YES

NO

rate=0.09

rate=0.12

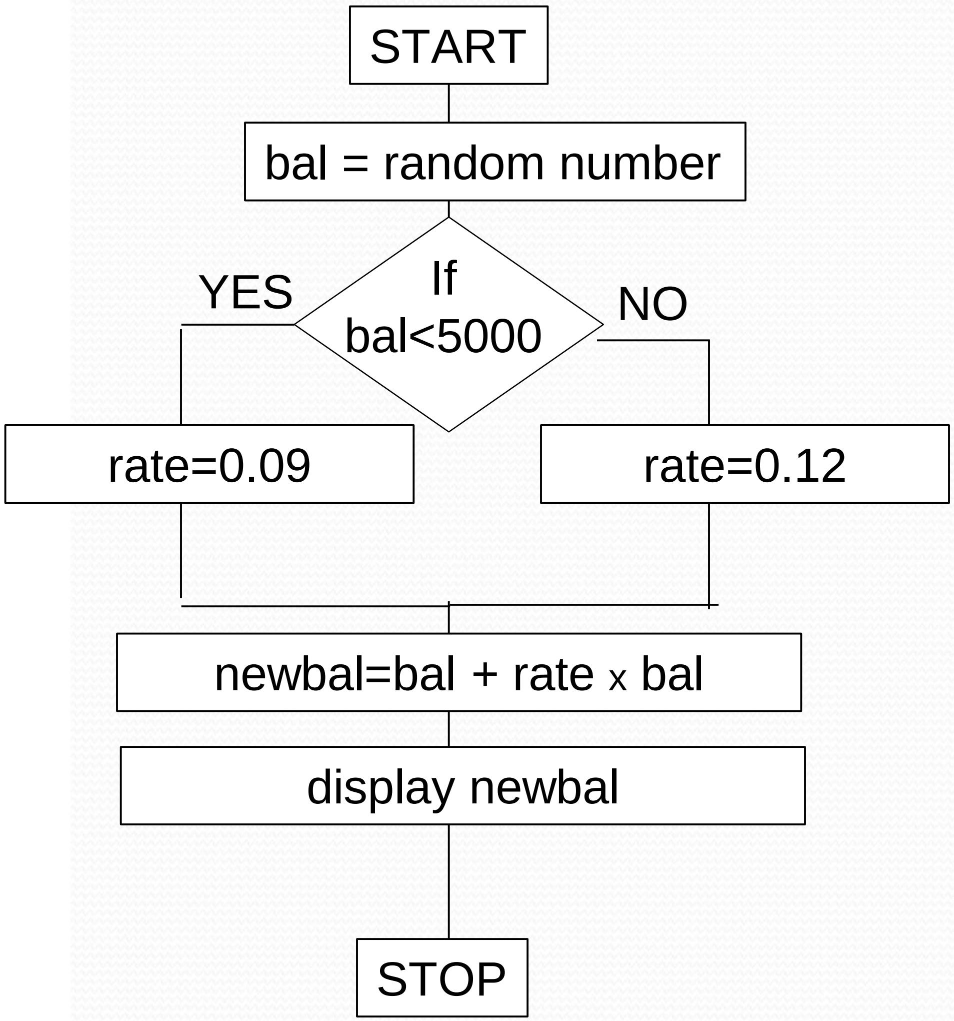
newbal=bal + rate x bal

display newbal

STOP

8. Decisions

Algorithm



START

bal = random number

If bal<5000

YES

NO

rate=0.09

rate=0.12

newbal=bal + rate x bal

display newbal

STOP

Program Matlab code

- bal = 10 000 * rand
- if bal < 5 000
  - rate = 0.09
- else
  - rate = 0.12
- end
- newbal = bal + rate * bal;
- disp(‘New balance after interest compounded is: ‘)
- format bank
- disp(newbal)
- format

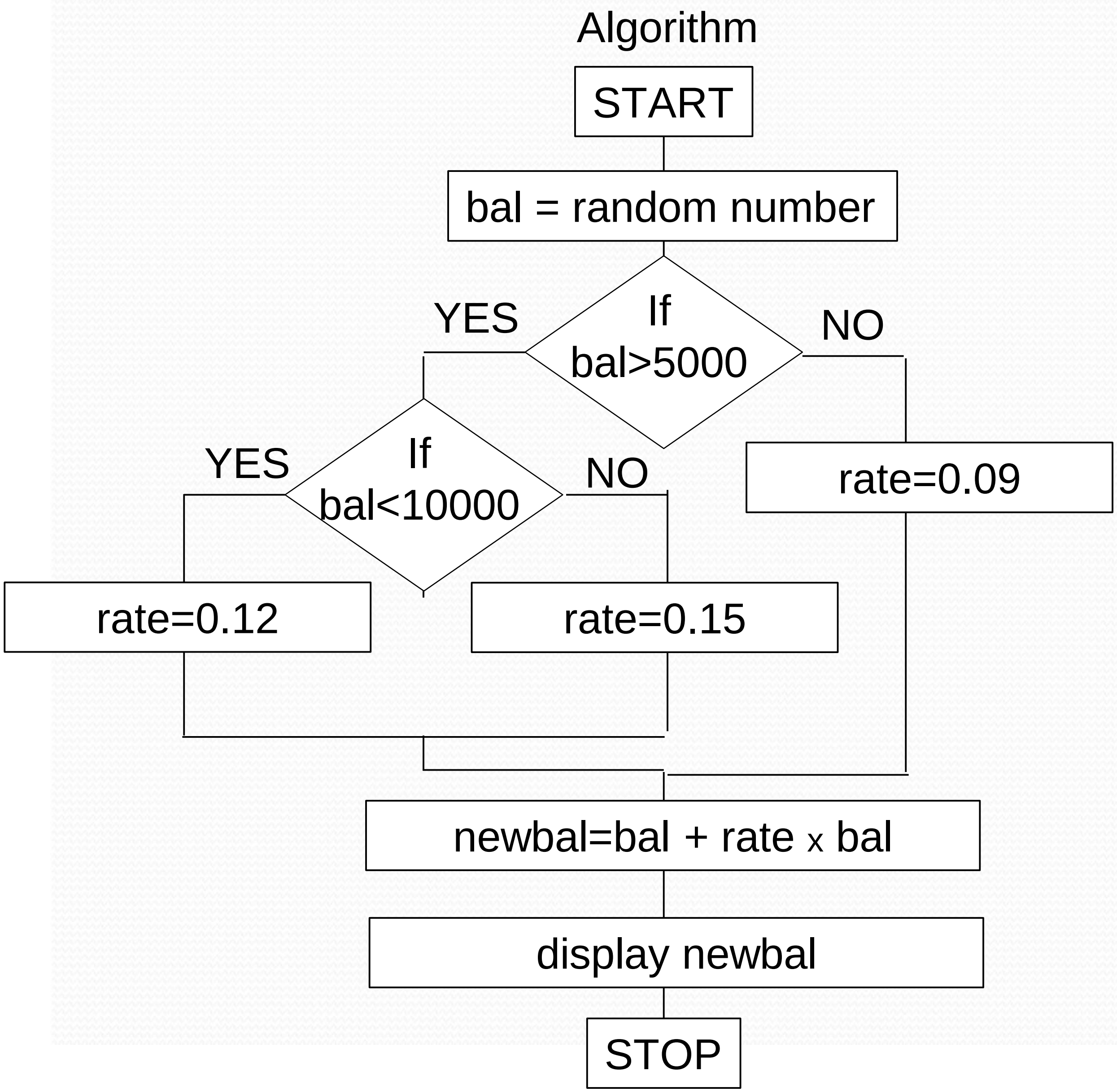
8. Decisions

- The one-line *if-else* statement
  - The simplest general form of *if-else* for use on one line is
    - if condition statements A, else statements B, end
  - Note
    - Commas (of semicolons) are essential between the various clauses.
    - The *else* part is optional.
    - Do not forget *end*, or MATLAB will wait forever.

8. Decisions

- Suppose the Random Bank now offers 9% on balances of less than $5000, 12% for balances of $5000 or more but less than $10000, and 15% for balances of $10000 or more.
- The following program calculates a customer's new balance after one year according to this scheme.

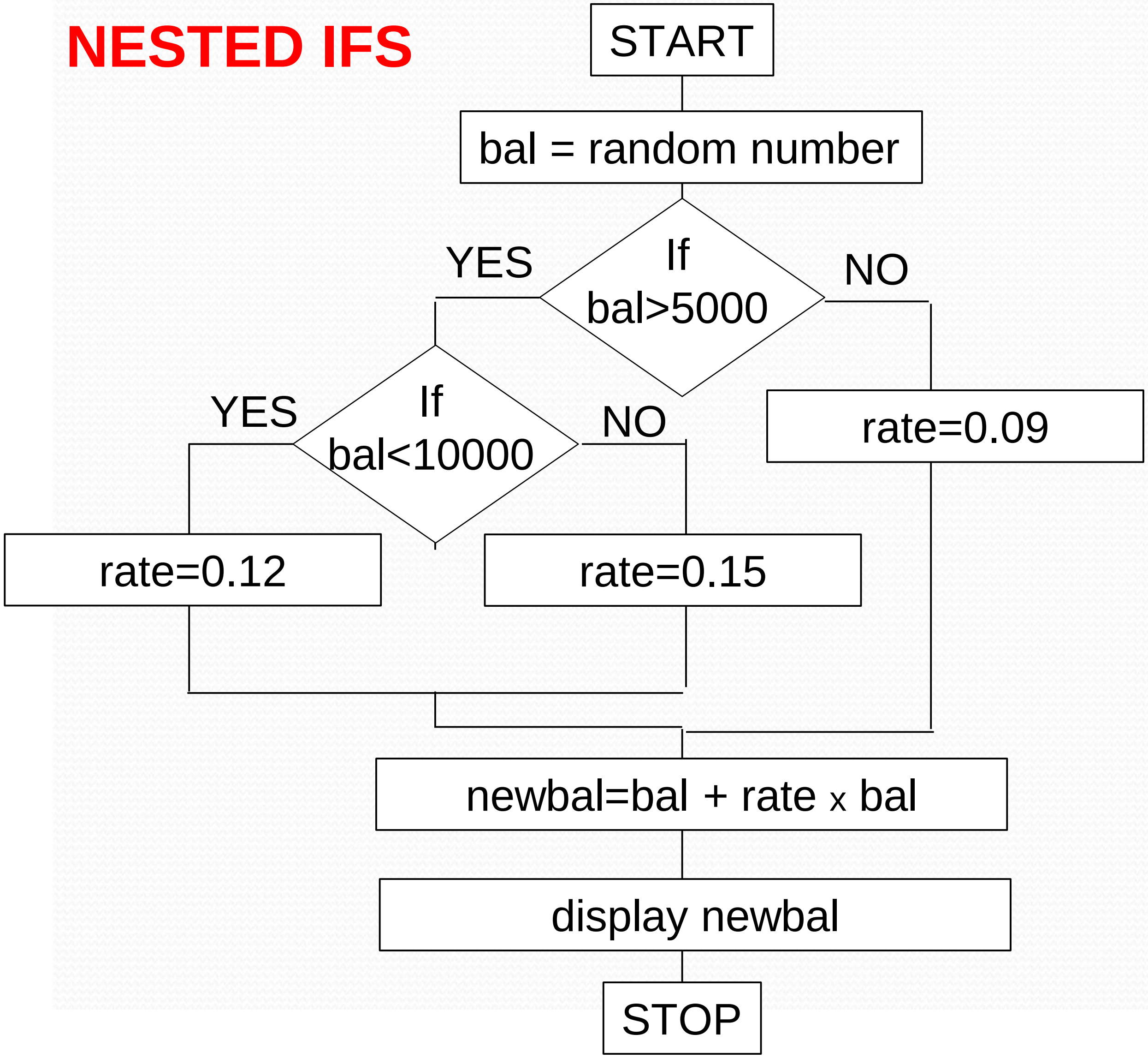
8. Decisions


Algorithm
START
bal = random number
If bal>5000
YES
NO
If bal<10000
YES
NO
rate=0.09
rate=0.12
rate=0.15
newbal=bal + rate x bal
display newbal
STOP

8. Decisions

NESTED IFS

Algorithm



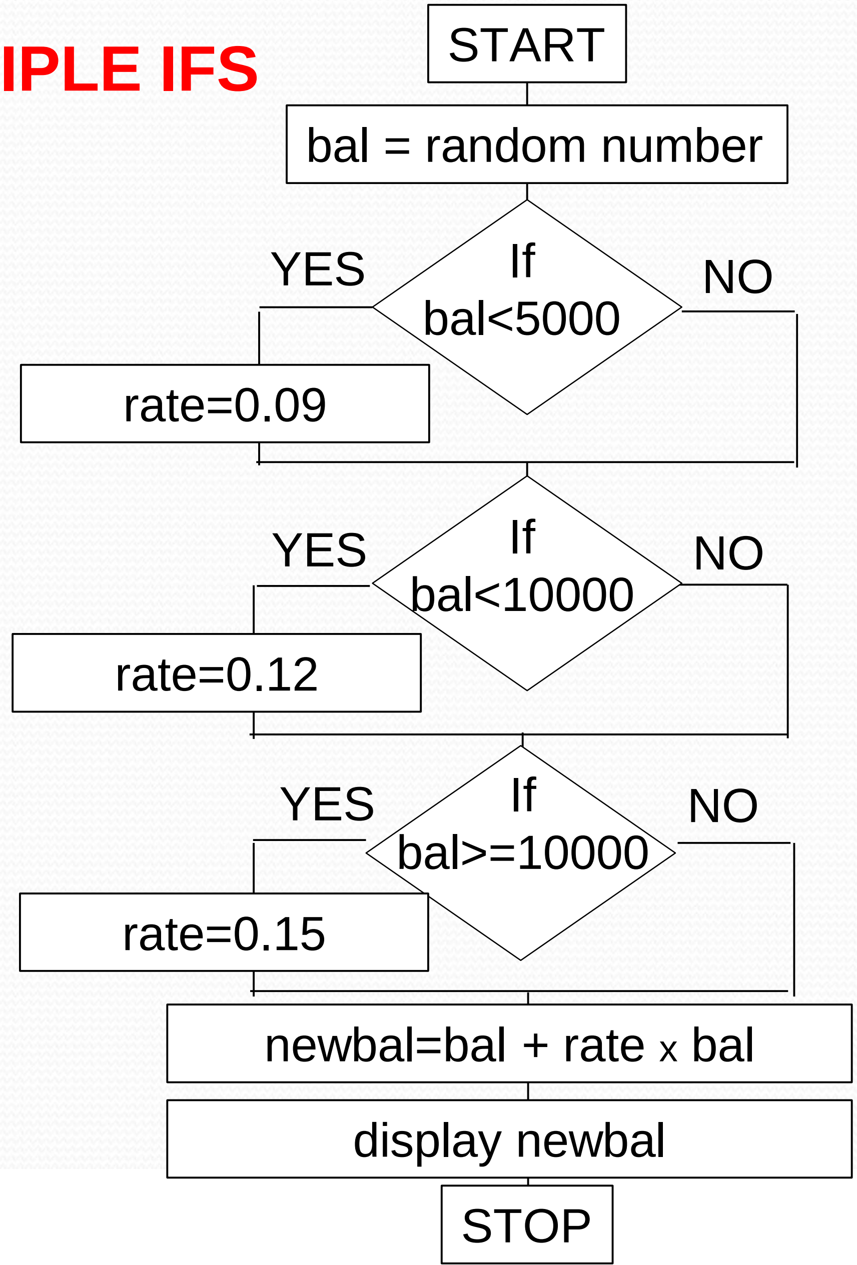
Program Matlab code

```
bal = 15000 * rand;
if bal > 5000
        if bal < 10000
          rate = 0.09;
        else
          rate=0.15
        end
else
  rate=0.09
end
newbal = bal + rate * bal;
format bank
disp( ‘New balance is: ‘ )
disp( newbal )
```

8. Decisions

MULTIPLE IFS

Algorithm



Program Matlab code

```
if bal < 5000
    rate = 0.09;
end
if bal < 10000
    rate = 0.12;
end
if bal >= 10000
    rate = 0.15;
end
  disp( 'New balance is: ' )
  disp( newbal )
```

8. Decisions

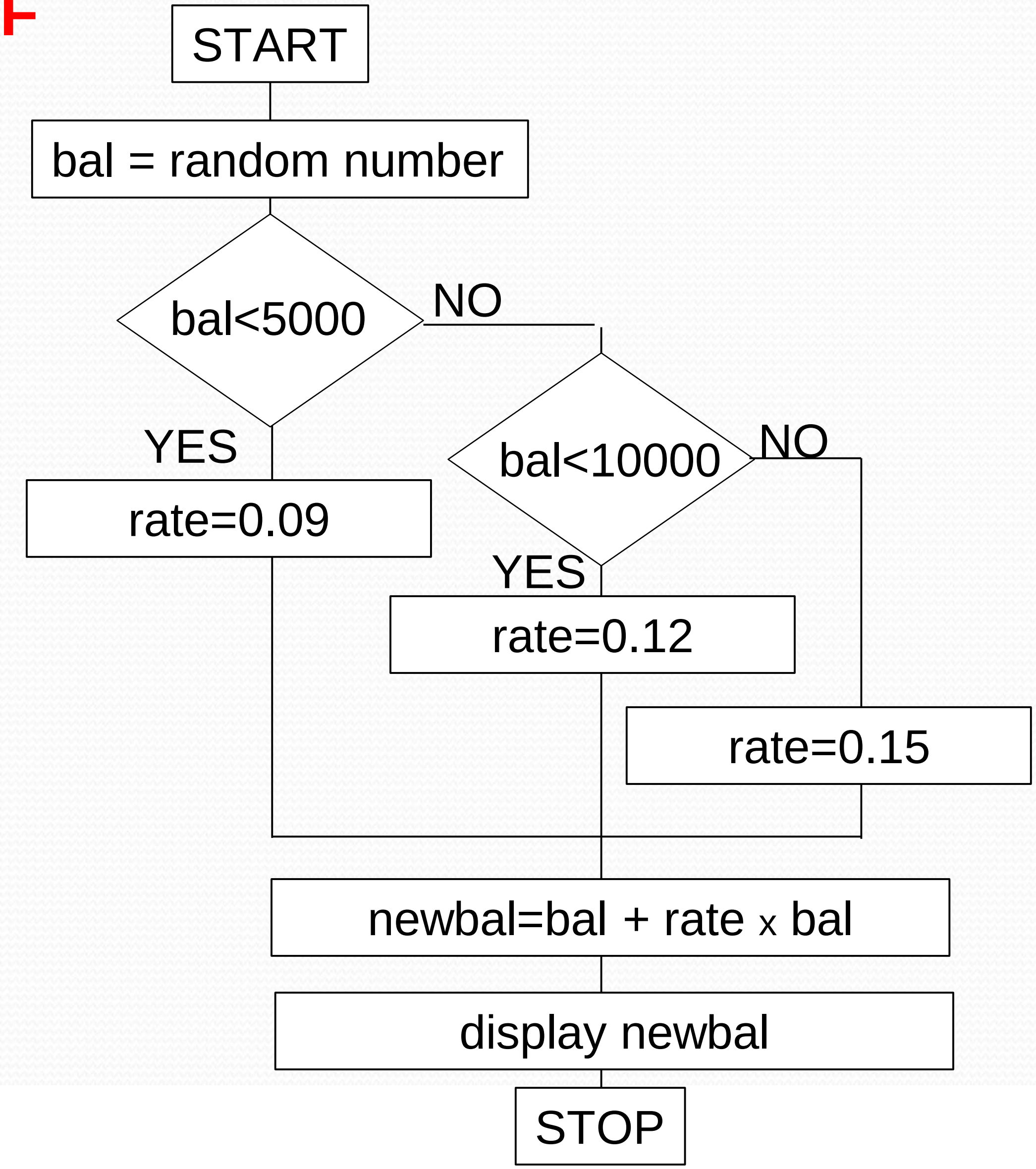
- Multiple *ifs* versus *elseif*
  - **Multiple *ifs* is inefficient** since each of the three conditions is always tested, even if the first one is true.
  - The mistake expressed in the following program is common. **You should always avoid it.**

- Nested ifs versus elseif
  - An *if* construct can contain further *ifs*, and so on. This is called *nesting* and should not be confused with *elseif* ladder.

8. Decisions

ELSEIF

Algorithm

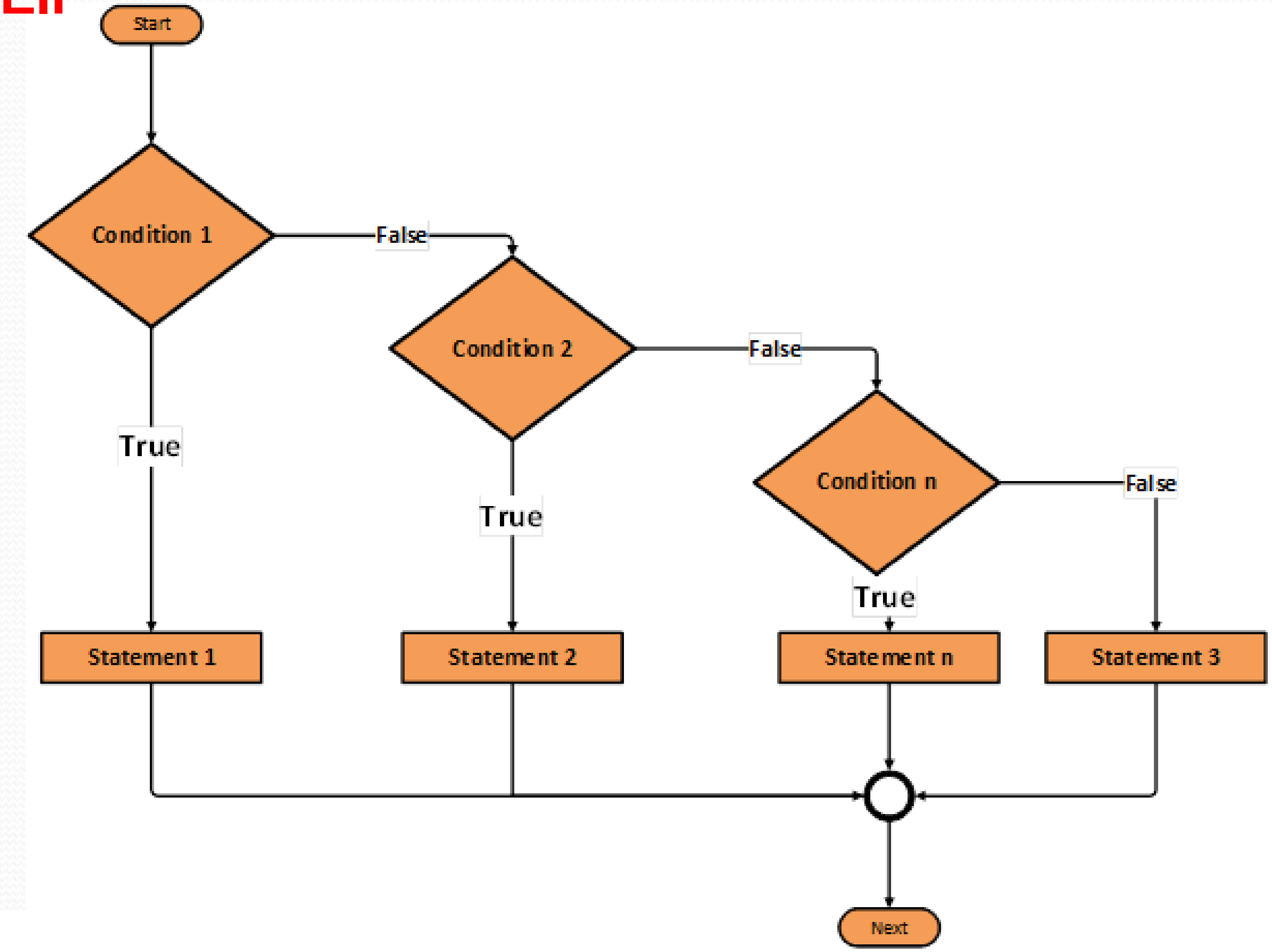


Program Matlab code

```
bal = 15000 * rand;
if bal < 5000
   rate = 0.09;
elseif bal < 10000
   rate = 0.12;
else
   rate = 0.15;
end
newbal = bal + rate * bal;
format bank
disp( ‘New balance is: ‘ )
disp( newbal )
```

8. Decisions

ELSEIF



8. Decisions

- elseif
  - Run the program a few times and display the values of *bal*, *rate*, and

    newbal to make sure MATLAB has chosen the correct interest rate.
  - In general, the *elseif* clause is used as follows:
    - if condition 1
      - statements A
    - elseif condition 2
      - statements B
    - elseif condition 3
      - statements C
    - …
    - else
      - statements E
    - end
  - That is sometimes called an elseif ladder

8. Decisions

elseif

- The elseif ladder works as follows

1. *condition 1* is tested. If it is true, then *statements A* are executed. MATLAB then moves to the next statement after *end*.
2. if *condition 1* is false, MATLAB checks *condition 2*. It if is true, *statements B* are executed, followed by the statement after *end*.

8. Decisions

elseif

- The elseif ladder works as follows

1. *condition 1* is tested. If it is true, then *statements A* are executed. MATLAB then moves to the next statement after *end*.
2. if *condition 1* is false, MATLAB checks *condition 2*. It if is true, *statements B* are executed, followed by the statement after *end*.
3. In this way, all the conditions are tested until a true condition is found. As soon as a true condition is found, no further *elseifs* are examined, and MATLAB jumps off the ladder.
4. If non of the conditions is true, *statements E* after *else* are executed.

8. Decisions

elseif

- The elseif ladder works as follows

1. *condition 1* is tested. If it is true, then *statements A* are executed. MATLAB then moves to the next statement after *end*.
2. if *condition 1* is false, MATLAB checks *condition 2*. It if is true, *statements B* are executed, followed by the statement after *end*.
3. In this way, all the conditions are tested until a true condition is found. As soon as a true condition is found, no further *elseifs* are examined, and MATLAB jumps off the ladder.
4. If non of the conditions is true, *statements E* after *else* are executed.
5. You should arrange the logic so that not more than one of the conditions is true.
6. There can be any number of *elseifs*, but at most one *else*.
7. *elseif* must be written as one word.
8. It is good programming style to *indent* each group of statements as it is done automatically by MATLAB and shown in this presentation.

8. Decisions

- Logical operators
  - More complicated logical expressions can be constructed using the three *logical operators* & (and), | (or), ~ (not).
  - Example: the quadratic $ax^2 + bx + c = 0$ has equal roots, given by , provided that b^2 -4$ac = 0$ and $a \neq 0$.

  This translates to the following MATLAB statements:

  - if (b^2 – 4*a*c == 0) & (a ~= 0)
    - x = -b / (2*a);
  - end
  - of course, a, b and c must be assigned values prior to reaching this set of statements. Note the double equal sign in the test for equality.

8. Decisions

Multiple *ifs* versus *elseif*

- Another mistake is common and illustrated in the following program.
  - Program
    - if 5000 <= bal < 10000
      - rate = 0.12;
    - end
  - In that case, whatever the value of *bal* is, this condition will always be true.

8. Decisions

- Nested *ifs*
    - Let us understand the concept of *nested ifs* through an example. Suppose you want to compute the solution of a quadratic equation. You may first want to check if $a = 0$, to prevent a division by zero.

8. Decisions

Nested *ifs*

- Program
  - d = b^2 – 4*a*c;
  - if a ~= 0
    - if d < 0
      - disp(‘Complex Roots’)
    - else
      - x1 = (-b + sqrt(d)) / (2*a) ;
      - x2 = (-b - sqrt(d)) / (2*a) ;
    - end
  - end

8. Decisions

- *ifs*
  - Pay attention to where you write the *end*.
  - The first *end* belongs to the second *if* by default, as intended.
  - Now, move the first *end* up as follows:
    - d = b^2 – 4*a*c;
    - if a ~= 0
      - if d < 0
        - disp(‘Complex Roots’)
      - end
    - else
      - x1 = (-b + sqrt(d)) / (2*a) ;
      - x2 = (-b - sqrt(d)) / (2*a) ;
    - end
  - Now, else belongs to the first *if*. Division by zero is therefore “forced” instead of prevented.

8. Decisions

- Switch
  - The *switch* statement executes certain statements based on the value of a variable or expression. In the following example, it is used to decide whether a random integer is 1, 2 or 3.
  - Program
    - d = floor(3*rand) + 1
    - switch d
      - case 1
        - disp(‘That’’s a 1!’);
      - case 2
        - disp(‘That’’s a 2!’);
      - case 3
        - disp(‘Must be 3!’);
    - end

8. Decisions

Switch

- Multiple expressions can be handled in a single case statement by enclosing the case expression in a cell array. Try the following program.
  - Program
    - d = floor(10*rand)
      - switch d
        - case { 2 , 4 , 6 , 8 }
          - disp(‘Even’);
        - case { 1 , 3 , 5 , 7 , 9 }
          - disp(‘Odd’);
        - otherwise
          - disp (‘Zero’);
    - end