CS 221 Lecture

Tuesday, 11 October 2011

"Computers in the future may weigh no more than 1.5 tons."

- Popular Mechanics, forecasting the relentless march of science, 1949.

Today’s Topics

1. Announcements
2. if statements (“logical statements” in the textbook) select among alternatives.
3. while repeats statements until a condition becomes false.
4. Formatted output is easier with fprintf().
5. Loops are useful for processing arrays element-by-element
6. for-loops: a shorthand for “bounded” loops

1. Announcements

- Remaining Quiz Dates:
  - In class: 25 October, 22 November
  - In lab: 3 November, 1 December
- Bring your text to lab!

2. if selects among alternatives.

```
if score >= 60
% score is at least 60
        grade = ’P’;   % this is alternative 1
else
% ~ (score >= 60)
% therefore: score < 60
        grade = ’F’;   % this is alternative 2
end
```

Exactly one alternative will be selected!

Quiz Problem – Correct Solution

```
if quality < 10
    disp(’Reject’)
elseif quality < 30
    % 10 ≤ quality < 30
    disp(’Maybe’)
else
    % quality ≥ 30 – no need to test!
    disp(’Accept’)
end
```

Quiz Problem – Common Mistakes

```
if quality < 10
    disp(’Reject’)
ifelse quality >= 10 && quality < 30
    disp(’Maybe’)
else  quality >= 30
    disp(’Accept’)
end
```

3. while Repeats Statements Until a Condition Becomes False

```
x = 10;
while x < 20
    x = x + 2
end
```

If the condition is initially false, the statement is never executed!

```
x = 30;
while x < 20
  x = x + 2   % this is not executed
end
```

Example: Euclid's Algorithm for the Greatest Common Divisor (GCD)

The Greatest Common Divisor (GCD) of two positive integers is the largest integer that divides both numbers.

- The GCD of two numbers is always ≥ 1
- Let's write GCD as a function:

  GCD(m,n) takes two positive integers and returns the largest integer that divides both m and n.
- The GCD function has the following properties:
  - GCD(x,x) == x
  - GCD(x,y) == GCD(x,x – y)

Euclid’s Algorithm Computes the GCD

Euclid’s algorithm*:

- Given two positive integers m and n:
  1. If m and n are equal, stop: m is the GCD (so is n).
  2. Otherwise (they are unequal):

     Replace the larger number with their difference
  3. Go back to the first step.

*Definition of algorithm: An effective procedure given as a sequence of steps for carrying out a specific computation.

Natural Language Description Corresponds to this MATLAB Code:

```
while  <m and n are not equal>
    <Replace the larger of m and n with the difference between them>;
end
```

GCD Function in MATLAB

```
function x = gcd(m,n)
% gcd: compute greatest common divisor
while m ~= n
    if m > n
        m = m – n;
    else  % n > m (Note: this is a COMMENT!)
        n = n – m;
    end
end
% at this point we know m==n
x = m;
end
```

4. Producing Formatted Output (Text Section 4.5)

- disp(): basic output capabilities
- Show a variable or array in default format
  - fixed number of decimal places
- What if you want to embed a number in a string?
  - E.g., to get “The number <v> is even.” where <v> is the value of variable v, you have to create an array of strings and convert v to a string with num2str():
  - disp([’The number ’, num2str(v), ’ is even’])
- What if you want to print only two decimal places?
- What if you don’t want a newlineline printed after the output?

fprintf() gives greater control over output formatting.

- fprintf(<format string>, var1, var2, ...)
  - <format string> is a string containing conversion indicators (starting with %) that show where to put the values of var1, var2, ... and how to format them
  - Example:

    fprintf('The value of x is %d\n', x) prints:

    The value of x is 100

    when x is 100.
  - Conversion indications consist of: % 12.5 d
    - %: indicates the beginning of the field
    - 12: minimum field width in characters
    - 5: precision (number of decimal places)
    - d: conversion to apply (d = decimal integer, i does the same thing)

fprintf Examples

5. Loops are useful for processing arrays element-by-element

You are given an array of numbers between 0 and 100. You want to print only the values in the array that are at least 70 and less than 90; all others should left blank.

For example:

```
V = [ 10  89   9  88  65  90  34  75  70]
```

should produce output:

```
V = [     89         88                    75  70]
```

Outlining a Solution

- Look at each element of the array:
  - If it is in the desired range, print it
    - Need each element to be the same width -> use fprintf()
  - Otherwise, print the appropriate number of blanks
- How to code this?
  - Need to process elements V(1), V(2), ... one at a time
  - Use a variable to hold the index into the array
    - Call the variable “i”
    - Start with i = 1 (smallest array index)
    - After processing each element, increase i by 1
    - Stop after processing the last element

How to find the max index of a vector?

length(V) returns the number of elements in V.

- For arbitrary array A: the largest dimension of A

Now we have:

```
i = 1;
while i <= length(V)
    <process element at index i>
    <increase i by 1>
end
```

Refining the Script

```
i = 1;
while i <= length(V)
    if <the $i^{th}$ element is in range>
        <print it with a space on either side>
    else
        <print 4 spaces>
    end
    <increase i by 1>
end
```

Refining the Script

- “i^{th} element is in range”

```
70 <= V(i) && V(i) < 90
```

- Print number V(i) with a space on either side:

```
fprintf(’ %2d ’, V(i))
```

- Print four spaces:

```
fprintf(’    ’)
```

Final Script?

```
i = 1;
while i <= length(V)
    if  70 <= V(i) && V(i) < 90
        fprintf(’ %2d ’, V(i))
    else
        fprintf(’    ’)
    end
    i = i + 1;
end
fprintf(‘\n’);
```

“Process the element at index i”

“increase i by 1”

Final Script

```
i = 1;
while i <= length(V)
    if  70 <= V(i) && V(i) < 90
        fprintf(’%6d’, V(i))
    else
        fprintf(’      ’)
    end
    i = i + 1;
end
fprintf(’\n’);
```

“Process the element at index i”

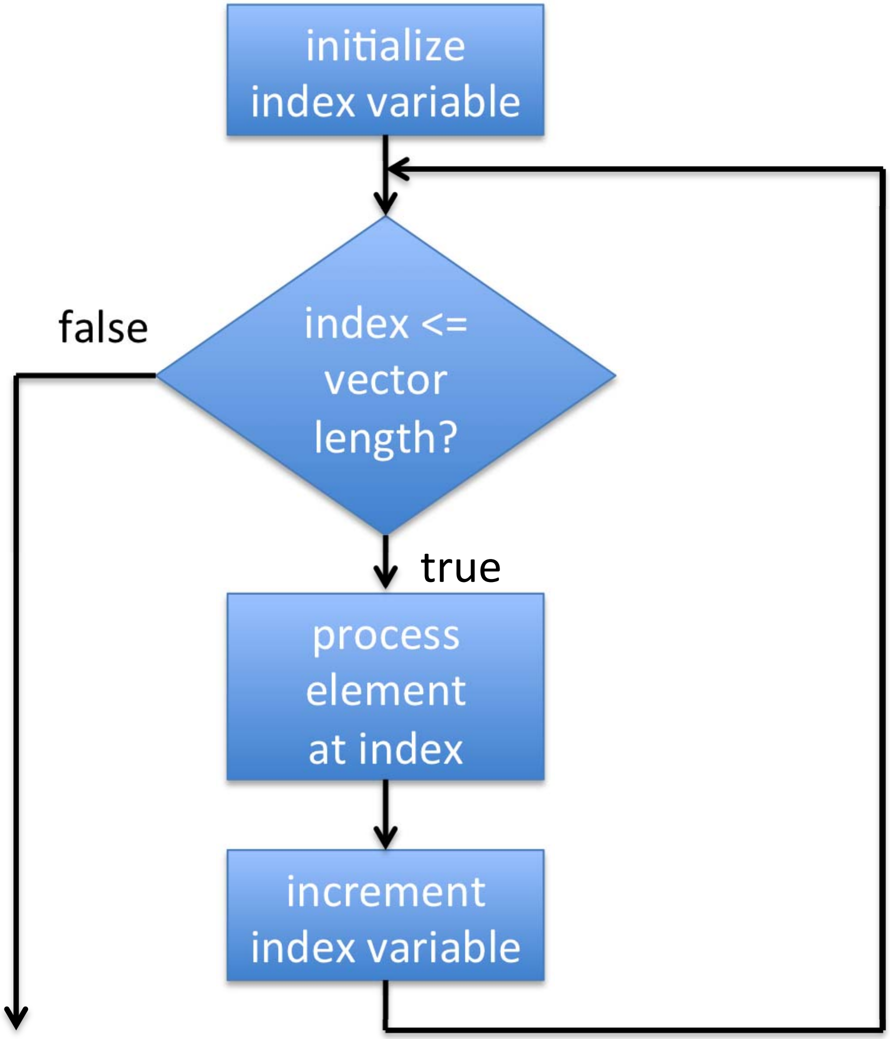
“increase i by 1”

Learn This Pattern!

- Iterating over the elements of a vector V using a while-loop:

```
i = 1;   % initialize index variable i
while i <= length(V)
    <do something with V(i)>
    i = i + 1;   % increment index!
end
```

Flowchart Pattern: Iteration over array with while



Iterating Over Some Elements

- Skip the first few elements:

```
i = 3;
while i <= length(V)
    …
    i = i + 1;
end
```

- Skip the last few elements:

```
i = 1;
while i <= length(V) – 3
    …
    i = i + 1;
end
```

Iterating Over Some Elements

- Every other element (odd indices only):

```
i = 1;
while i <= length(V)
    ...
    i = i + 2;
end
```

- Every other element (even indices only):

```
i = 2;
while i <= length(V)
    ...
    i = i + 2;
end
```

5. for-loops provide a shorthand for "bounded" loops

- MATLAB, like many programming languages, has a shorthand for this kind of loop:

```
for i=1:length(V)
    <statement>
end
```

- Read this as:

  "for each (integer) value from 1 to length(V), execute <statement> with i having that value"

- This is equivalent to the while-pattern just shown
  - MATLAB automatically initializes i to 1, tests for exceeding the maximum before, and increments i after <statement>
- Usually <statement> uses i as an index into V
  - But it is not required to do so

for-loops provide a shorthand for certain while-loops

```
for i=1:length(V)
    <statement>
end
```

- This is equivalent to the while-pattern seen earlier
  - MATLAB automatically initializes i to 1, tests for exceeding the maximum before, and increments i after <statement>
  - <statement> will be executed length(V) times
- Usually <statement> uses i as an index into V
  - But it is not required to do so

Example Script Revisited

```
% i = 1 not needed!
for i = 1:length(V)
    if   70 <= V(i) && V(i) < 90
         fprintf(’%6d’, V(i))
    else
         fprintf(’      ’)
    end
    % i = i + 1 not needed!
end
fprintf(’\n’);
```

The General Form of a for-loop

```
for <variable> = <vector expression>
    <statement>
end
```

<variable> can be any MATLAB variable name.

<vector expression> follows the pattern:

<start value> [: <increment>] : <end value>

The effect is to begin with <start value> and increase by <increment> until the value exceeds <end value>

If the <increment> is not included it is set to 1

for-loop examples

```
for index = 23:44
    ...
end
```

The loop is executed 22 times, with index having the values 23, 24, 25, ... , 43, 44

```
for k = 3:4:19
    ...
end
```

Here k takes on the values: 3, 7, 11, 15, 19

Problem: Counting elements in a vector

- Write a function “inrange()” that takes three arguments:
  - a vector (of any size)
  - a lower bound
  - an upper bound

  ... and returns the number of elements in the vector that are between the bounds, i.e., that are at least the lower bound and less than the upper bound
- Use a for-loop to iterate over the elements

Counting Elements with Some Property

```
function count = inrange(V, lower, upper)
% inrange: count elements of vector between bounds
    count = 0;
    for j=1:length(V)
        if V(j) >= lower && V(j) < upper
            count = count + 1;
        end
    end
end
```

Iterating Over 2-Dimensional Arrays Requires Nested Loops

- How can we process each element of a two-dimensional array?
  - Elements are accessed via two indices: A(row,col)
- Example: A is a 3 x 5 matrix
  - We need all 15 combinations of row and column #s:

(1,1) (1,2) (1,3) (1,4) (1,5)
(2,1) (2,2) (2,3) (2,4) (2,5)
(3,1) (3,2) (3,3) (3,4) (3,5)

Example: Summing positive elements in a 3x5 array

```
sum = 0;  % to hold the sum
% first row
for col=1:5
   if A(1,col) > 0
        sum = sum + A(1,col);
   end
end
% second row
for col=1:5
   if A(2,col) > 0
        sum = sum + A(2,col);
   end
end
% third row
for col=1:5
   if A(3,col) > 0
        sum = sum + A(3,col);
   end
end
```

Example: Summing positive elements in a 3x5 array

```
sum = 0;  % to hold the sum
row = 1;  % first row
for col=1:5
   if A(row,col) > 0
        sum = sum + A(row,col);
   end
end
row = 2;  % second row
for col=1:5
   if A(row,col) > 0
        sum = sum + A(row,col);
   end
end
row = 3;  % third row
for col=1:5
   if A(row,col) > 0
        sum = sum + A(row,col);
   end
end
```