



WICHITA STATE  
UNIVERSITY  
COLLEGE OF ENGINEERING  
*Biomedical Engineering*

# Good Programming Practices

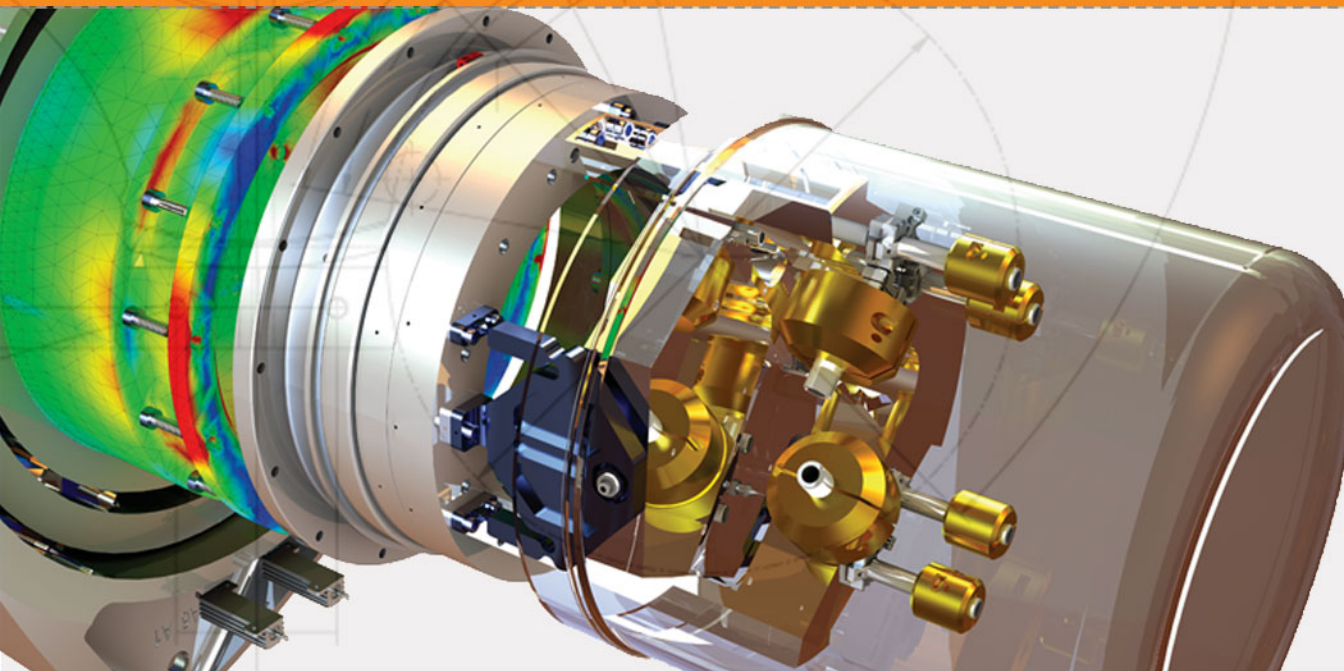


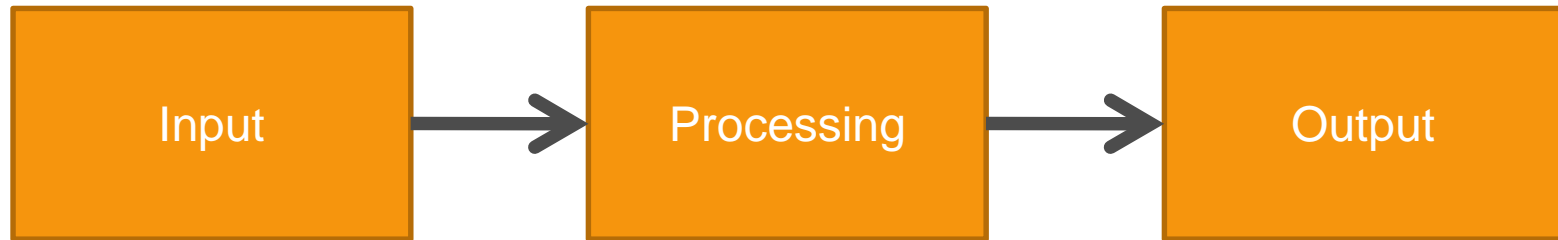
Image courtesy of National Optical Astronomy  
Observatory, operated by the Association of Universities  
for Research in Astronomy, under cooperative  
agreement with the National Science Foundation.

# Overview

- Program operation and program planning
- Planning tools
  - Flowcharts
  - Pseudocode
- Commenting
- Formatting

# Program Operation

- Typical order of sequence for program operation



- When designing your program, think about these steps in the following order:
  - Output: what do I want my program to produce/create/do?
  - Input: what information do I have for my program to work with?
  - Processing: how do I take the input information that I have and get the output information I want?

# Looking for Repeats

- Problem: How fast is a car traveling if it goes 50 miles in 2 hours?
- Output – speed
- Input – mileage covered, time elapsed
- Processing – basic physics:

$$\text{speed} = \frac{\text{miles}}{\text{time}}$$

- Speed = 50miles/2hours = 25 mph

# Example

- Problem: A cattle train left Miami and traveled toward New York. 14 hours later a diesel train left traveling at 45 km/h in an effort to catch up to the cattle train. After traveling for four hours the diesel train finally caught up. What was the cattle train's average speed?
- Output – Cattle train's average speed
- Input – diesel train's speed (45km/h), time between departures (14h), time to catch up (4h)
- Processing – basic algebra

$$\text{distance} = \text{speed}_{\text{diesel}} \times \text{time}_{\text{catch-up}}$$

$$\text{speed}_{\text{cattle}} = \frac{\text{distance}}{\text{time}_{\text{catch-up}} + \text{time}_{\text{between}}}$$

$$\text{speed}_{\text{cattle}} = \frac{180\text{km}}{4\text{h} + 14\text{h}} = 10\text{km/h}$$

# Functions

- Used for repeated code

*function variable = Name (input1, input2,...)*

- *Example, multiply two numbers:*

```
function z = my_multiply(x, y)
```

```
z = x * y;
```

- To execute or repeat the code you “call” it with values:

```
value = my_multiply(2,9)
```



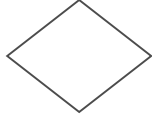

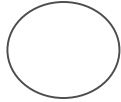

- Value variable will be 18

# Flow Charts

- A flow chart is a graphical map or visual plan of the logic needed to solve the given problem or implement an algorithm.
- A flow chart does not require any program code and therefore allows the programmer to focus only on the logic or plan of action without worrying about syntax.
- A good flow chart will save the programmer time when writing code and can be very helpful if debugging becomes necessary.

# Flow Charts

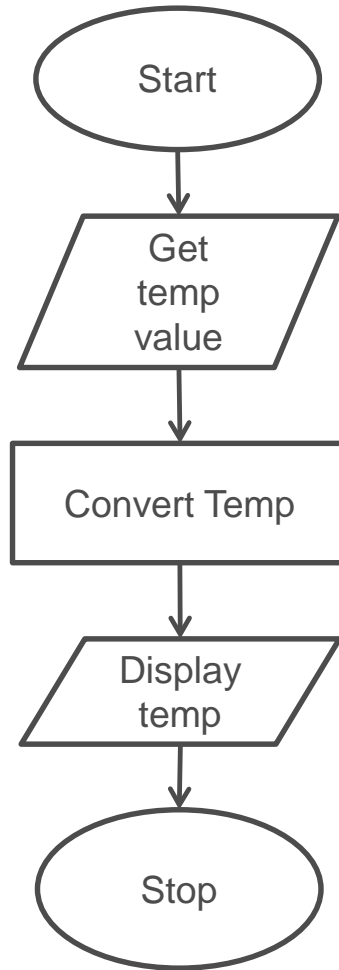
- Standard Flowcharting Symbols:

- Start/Stop.....
- Process instruction.....
- Conditional test (decision or loop).....
- Input/Output.....
- Connector (to another place).....
- Flowline.....



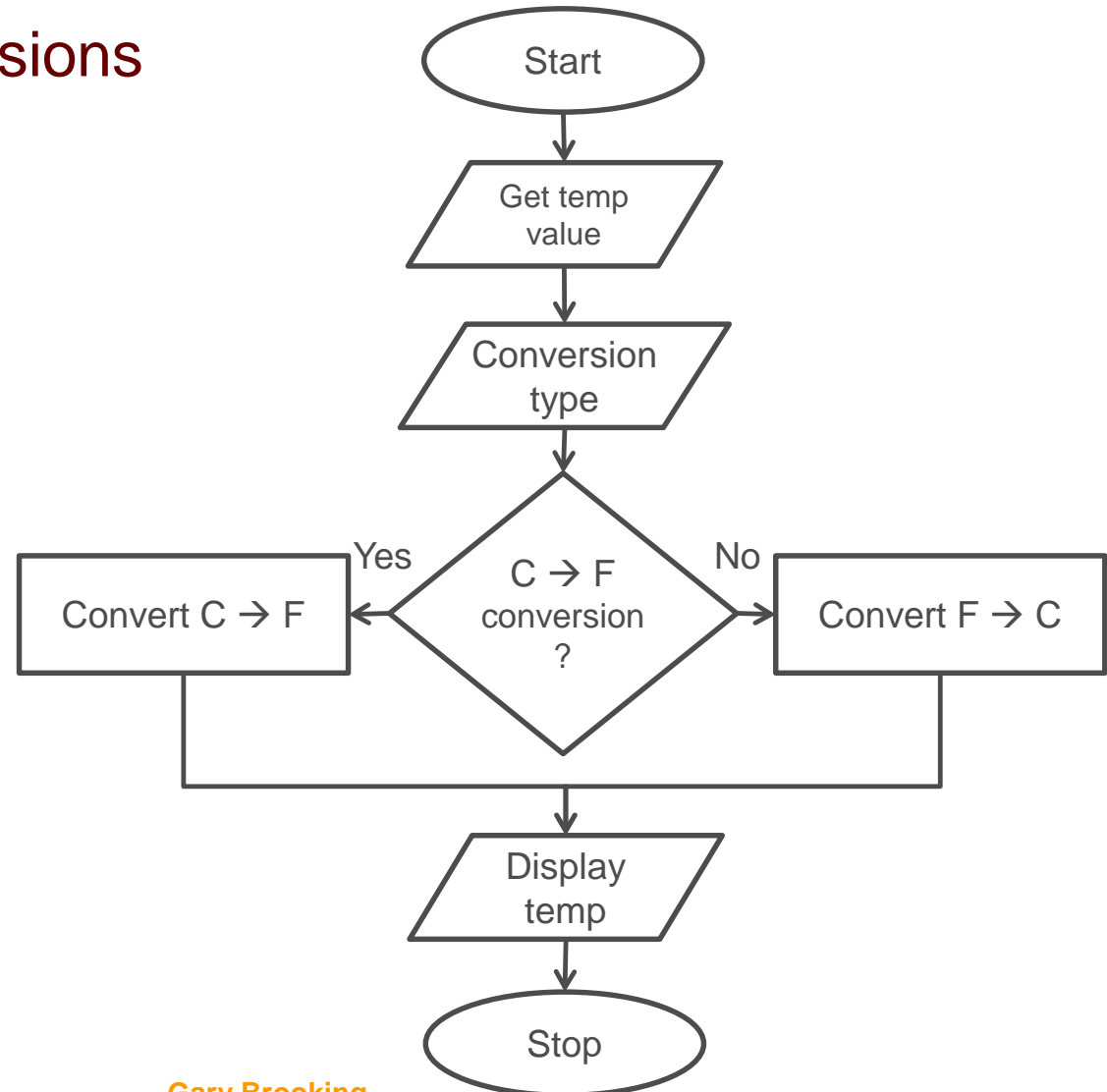
# Example

- Linear flow chart



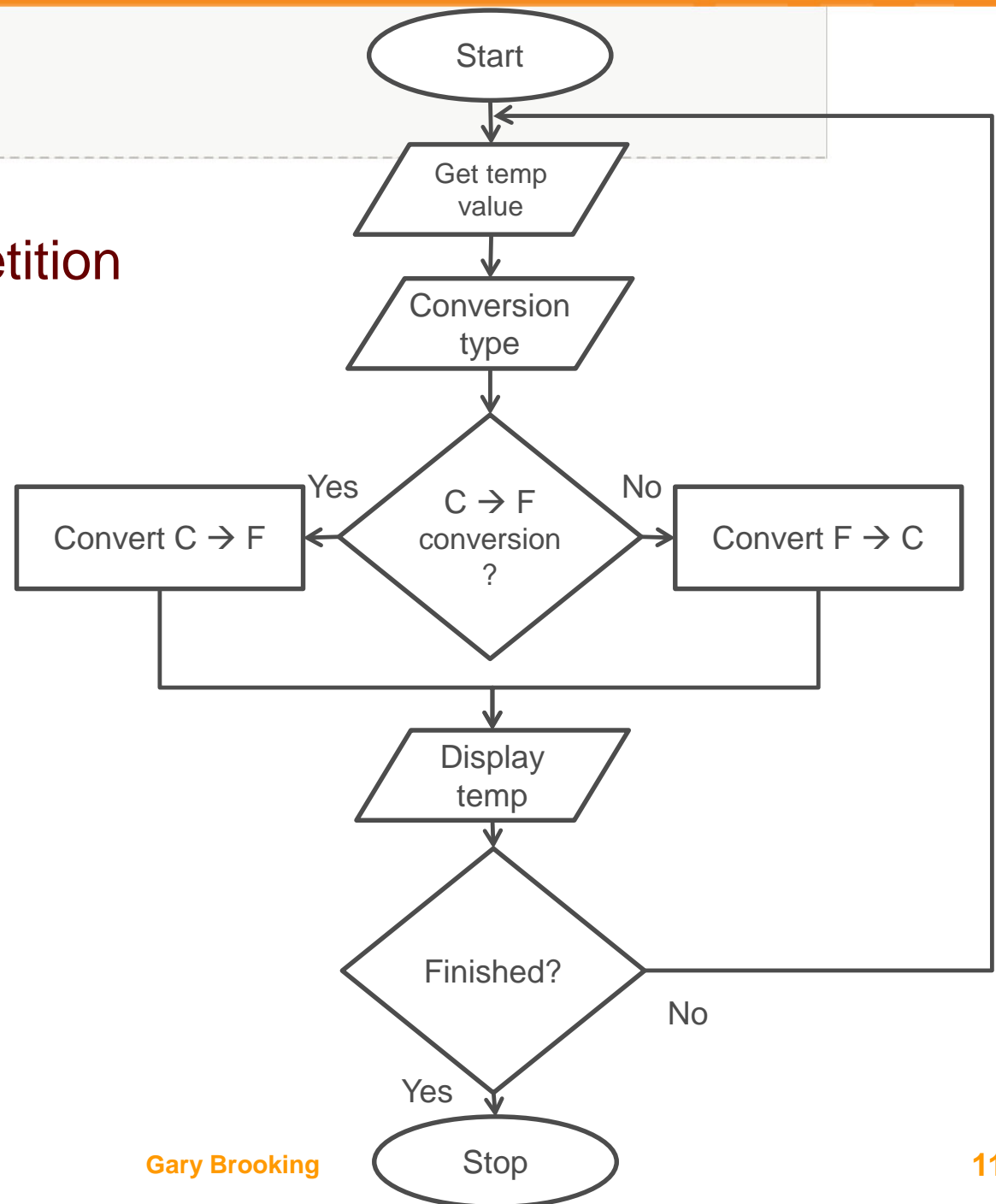
# Example

- Flow chart with decisions



# Example

- Flow chart with repetition



# Pseudocode

- Textual representation of the logical steps required to carry out a given task
- Pseudocode can:
  - have statements similar to programming statements
  - have structure similar to actual programs (i.e indents, white space)
  - be less specific than an actual program

# Examples

- Linear Pseudocode

Get temperature input from user

Convert temperature value

Display temperature value

# Examples

- Pseudocode with decisions

Get temperature input from user

Get conversion type from user

IF conversion type is C  $\rightarrow$  F

    Convert temperature value from C  $\rightarrow$  F

OTHERWISE

    Convert temperature value from F  $\rightarrow$  C

Display temperature value

# Examples

- Pseudocode with repetition

WHILE not finished

    Get temperature input from user

    Get conversion type from user

    IF conversion type is C  $\rightarrow$  F

        Convert temperature value from C  $\rightarrow$  F

    OTHERWISE

        Convert temperature value from F  $\rightarrow$  C

    Display temperature value

    Ask user if finished

# Pick Variable Names that Mean Something

Picking generic variable names makes code very difficult to read or debug or revise.

For example:

```
if q >= 10
    s = 0.9*p;
    disp('s='), disp(s);
else
    disp('s='), disp(p);
end
```

What does this mean?

How about this?

```
if quantity >= 10
    salePrice = 0.9*price;
    disp('itemPrice='), disp(salePrice);
else
    disp('itemPrice='), disp(price);
end
```



# Comments

- A comment is a line of code that conveys information about your program to someone reading it, but is not executed when the program is run
  - Comments allow others to view your code and understand what you did
  - Comments allow you to remember what you did!
- To create a comment in MATLAB use the % symbol
  - Comments will appear in green
  - Comments can be their own lines or can be placed after executable lines

# Comments

- Code submission header:
  - Whenever you submit code for an assignment, you must include the following header at the top of your code:

```
% name_of_code_file.m – Your name(s) – date completed  
% Description: describe what your code does  
% Usage: describe how to run your code (i.e. what  
% format any inputs need to be given in, are there input  
% arguments, etc)
```

# Comments

- Code submission header example:
  - A MATLAB function which requires the user to enter two numbers and which returns the product of the two numbers

```
% my_multiply.m – Dr. B – 06/01/2015  
% Description: this function allows the user to enter two  
% numbers and the program returns the product  
% Usage: product = my_multiply(num1, num2);  
% where num1 and num2 are two numbers and product  
% is where the result of the multiplication will be stored
```

# Use Good Format

Using good format will make your program much easier to read:

1. Use whitespace (blank lines or spacing within lines) to make the program more readable. For example, leaving space around arithmetic operators make them stand out.
2. Use proper indenting so that loops and conditional statements are easily readable and it is easy to determine what pieces of code are nested within other pieces of code.
3. Use comment lines to help break up sections so it is easier to locate sections of code.

# Test Your Program

Never assume that if your program executes with no errors that the program is working perfectly fine. Always test your program with a variety of inputs to make sure that the program is producing the correct results.

Make allowances for bad user input to ensure your program will not terminate unexpectedly or give incorrect results.