- Algorithm: an ordered sequence of precisely defined instructions that performs some task a finite amount of time.
- Ordered means that the instructions can be numbered, but an algorithm must have the ability to alter the order of its instructions using a control structure.
- There are three categories of algorithmic operations:
  - Sequential Operations: Instructions executed in order.
  - Conditional Operations: Control structures that first ask a question to be answered with a true/false answer and then select the next instruction based on the answer.
  - Iterative Operations (Loops): Control structures that repeat the execution of a block of instructions.

Program Design and Development

Structured Programing: is a programming paradigm aimed at improving the clarity, quality, and development time of a computer program by making extensive use of the structured control flow constructs of selection (if/then/else) and repetition (while and for), block structures, and subroutines.

Advantages of Structured Programming:

1. Structured programs are easier to write because the programmer can study the overall problem first and then deal with the details later.
2. Modules (functions) written for one application can be used for other applications (this is called reusable code).
3. Structured programs are easier to debug because each module is designed to perform just one task and thus it can be tested separately from the other modules.
4. Structured programming is effective in a teamwork environment because several people can work on a common program, each person developing one or more modules.
5. Structured programs are easier to understand and modify, especially if meaningful names are chosen for the modules and if the documentation clearly identifies the module's task.

Steps for Creating Computer Solutions:

1. State the problem concisely.
2. Specify the data to be used by the program. This is the "input".
3. Specify the information to be generated by the program. This is the "output".
4. Work through the solution steps by hand or with a calculator; use a simpler set of data if necessary.
5. Write and run the program.
6. Check the output of the program with your hand solution.
7. Run the program with your input data and perform a reality check on the output.

8. If you will use the program as a general tool in the future, test it by running it for a range of reasonable data values; perform a reality check on the results.

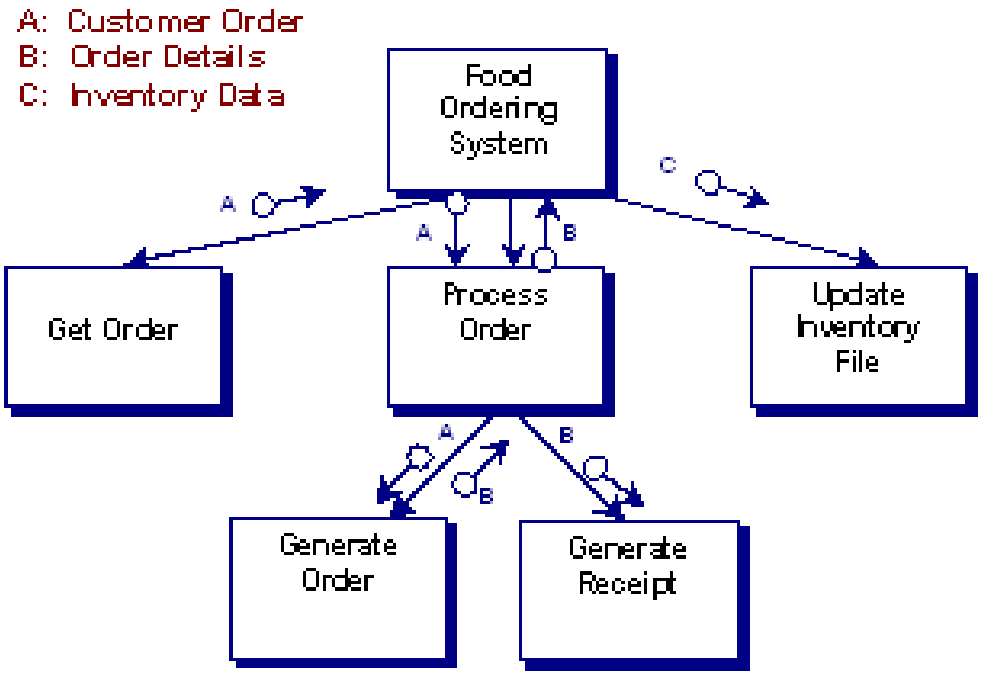
Documentation:

1. Proper selection of variable names to reflect the quantities they represent.
2. Use of comments within the program.
3. Use of structure charts.
4. Use of flowcharts.
5. A verbal description of the program, often in pseudocode.

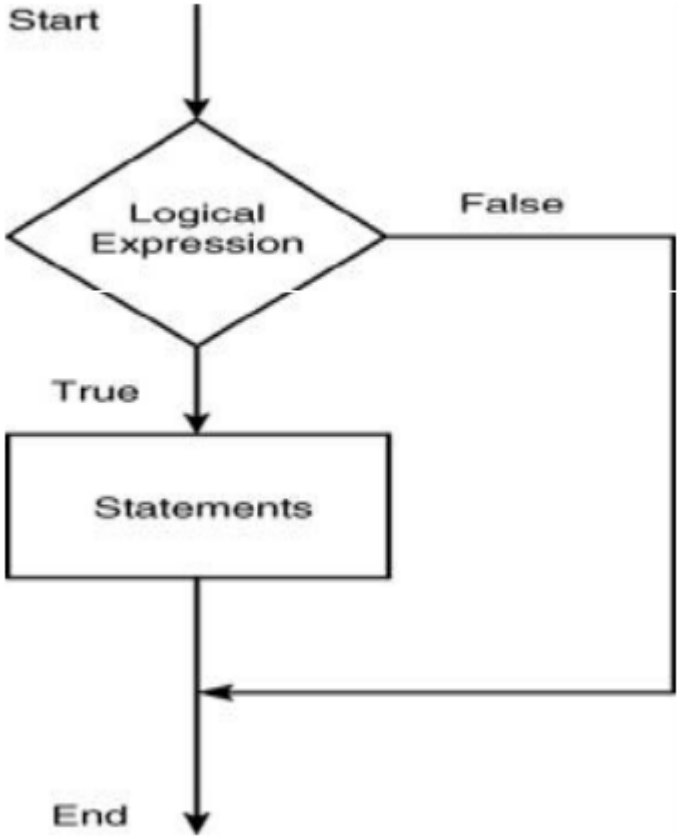
Documenting with Charts:

- Two types of charts aid in developing structured programs and in documenting them. These are structure charts and flowcharts.

- A structure chart is a graphical description showing how the different parts of the program are connected.



- Flowcharts are useful for developing and documenting programs that contain conditional statements, because they can display the various paths (called "branches") that a program can take, depending on how the conditional statements are executed.



Documenting with Pseudocode:

- We can document with pseudocode, in which natural language and mathematical expressions are used to construct statements that look like computer statements but without detailed syntax.
- Each pseudocode instruction may be numbered but should be unambiguous and computable.

Relational Operators in Matlab:

- Six operators that are used for comparison of variables (scalars or arrays) or expressions.
- They have equal precedence (evaluated from left to right).

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

- For example, suppose that x = [6,3,9] and y = [14,2,9]. The following Matlab session shows some examples.
- Note: Comparison is performed element-by-element.

```
>> x = [6 3 9];
>> y = [14 2 9];
>> z = x<y

z =

  1×3 logical array

   1   0   0

>> z = x~=y

z =

  1×3 logical array

   1   1   0

>> z = x>8

z =

  1×3 logical array
```

```
0   0   1
```

- The relational operators can be used for array addressing. For example, with x = [6,3,9] and y = [14,2,9], typing z = x(x<y)
  finds all the elements in x that are less than the corresponding elements in y. The result is z=6.
- The arithmetic operators +, -, *, /, and \ have precedence over the relational operators. Thus the statement z = 5 > 2 + 7 is equivalent to z = 5 >(2+7) and returns the result z = 0.
- We can use parentheses to change the order of precedence; for example, z = (5 > 2) + 7evaluates to z = 8.

The Logical Class:

- When the relational operators are used, such as x = (5 > 2) they create a logical variable, in this case, x.
- Logical variables may have only the values 1 (true) and 0 (false).

Logical Operators:

Operator	Name	Definition

~	NOT	~A returns an array with the same dimensions as A; the new array has ones where A is zero and zeros where A is non-zero.
		`>> A = [1 2 0 4 0 7];`
		`>> C = ~A`
		`C =`
		`1×6 logical array`
		`0 0 1 0 1 0`

&	AND	A&B returns an array with the same dimensions as A and B; the new array has ones where both A and B have non-zero elements, and zeros where either A or B is zero.
		`>> A = [1 2 0 4 0 7];`
		`>> B = [0 1 8 2 0 3];`
		`>> A&B`
		`ans =`
		`1×6 logical array`
		`0 1 0 1 0 1`

|	OR	A|B returns an array with the same dimensions as A and B; the new array has ones where at least on element in A or B is non-zero, and zeros where A and B are both zero.
		`>> A = [1 2 0 4 0 7];`
		`>> B = [0 1 8 2 0 3];`
		`>> A&B`
		`ans =`

		1×6 logical array 1 1 1 1 0 1
&&	Short-Circuit AND	Same as AND but used with scalars
||	Short-Circuit OR	Same as OR but used with scalars

Precedence of Different Operators:

Precedence	Operator type
First	Parentheses; evaluated starting with the innermost pair.
Second	Arithmetic operators and logical NOT (~); evaluated from left to right.
Third	Relational operators; evaluated from left to right.
Fourth	Logical AND.
Fifth	Logical OR.

Logical Functions:

Logical Function	Definition
ischar(A)	returns 1 if A is a character array and 0 otherwise
isempty(A)	returns 1 if A is an empty matrix and 0 otherwise
isinf(A)	returns an array with the same dimension as A with ones where A has inf and zeros eksewhere.
isnan(A)	returns an array with the same dimension as A with ones where A has NaN and zeros eksewhere.
isnumeric(A)	returns 1 if A is a numeric array and 0 otherwise
isreal(A)	returns 1 if A has no elements with imaginary parts and 0 otherwise
logical(A)	converts the elements of array A into logical values

- Use Matlab Help to check out the following logical functions:
  Isstr
  Isvector
  Ishold
  Isequal
  Isscalar

The Find Function:

- The Matlab function (find) computes an array containing the indices of the nonzero elements of the numeric array x.
  >> x = [-2 0 4];
  ```
  >> y = find(x)
  y =

  1     3
  ```
  The resulting array y = [1, 3] indicates that the first and third elements of x are nonzero.
- [u,v,w] = find(A)
  Computes the arrays u and v containing the row and column indices of the nonzero elements of the array A and computes the array w containing the values of the nonzero elements. The array w may be omitted.
- When the find function is used with arrays, the return value is the linear indices of the nonzero elements.
- With linear indices, we can refer to array elements with a single value A(k), where k is the element location if the columns of the array are appended to each other.

$$A = \begin{bmatrix} -2 & -1 & 0 \\ 5 & 6 & 7 \\ 1 & 2 & 3 \end{bmatrix}$$

  A(2) is equivalent to A(2,1) which is 5
  A(5) is equivalent to A(2,2) which is 6
- When the Find Function is used with logical operators, the expression inside the function is evaluated first, then the search for nonzero elements is performed.

```
>> x = [5 -3 0 0 8];
>> y = [2 4 0 5 7 ];
>> z = find(x&y)

z =

     1     2     5
```

Note that the find function returns the indices, and not the values. To extract the values, use array indexing w = x(z).

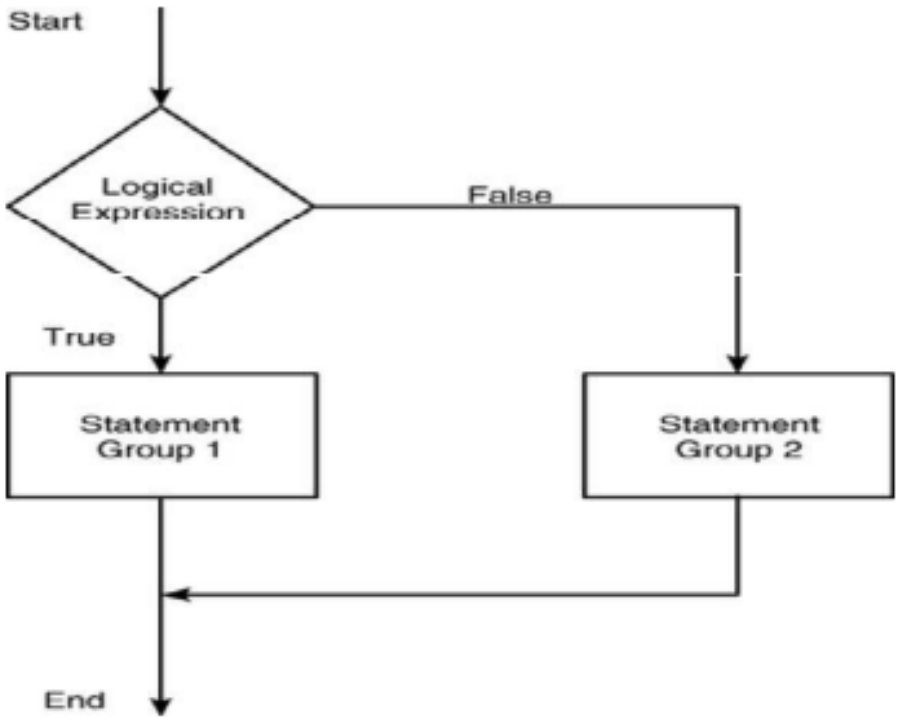
Conditional Statements

- The if statement's basic form is:

```
if  logical expression
        statements
end
```

- Every if statement must have an accompanying end statement.
- The end statement marks the end of the statements that are to be executed if the logical expression is true.
- The basic structure for the use of the else statement is:

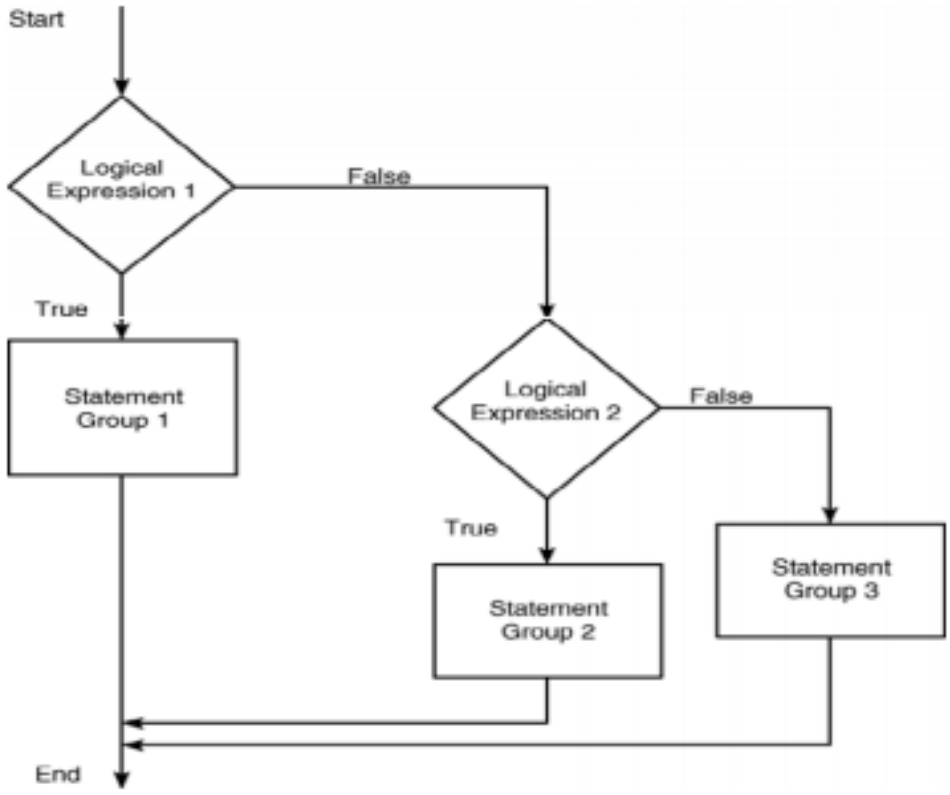
```
if      logical expression
        statements group 1
else
        statement group 2
end
```



- The general form of the if statement is:

```
if  logical expression 1
        statement group 1
elseif  logical expression 2
        statement group 2
else
        statement group 3
end
```

The else and elseif statements may be omitted if not required. However, if both are used, the else statement must come after the elseif statement to take care of all conditions that might be unaccounted for.



Example:

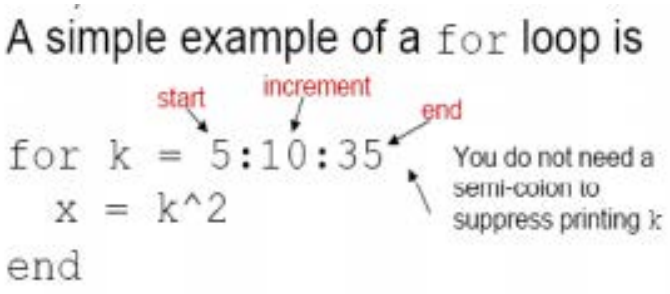
$$y = \begin{cases} \exp(x) - 1, & x < 0 \\ \text{sqrt}(x), & 0 \le x \le 10 \\ \log(x), & x > 10 \end{cases}$$

```
>> x = [-3:0.1:11];
>> if x > 10
y = log(x);
elseif x >= 0
y = sqrt(x);
else
y = exp(x)-1;
end
```

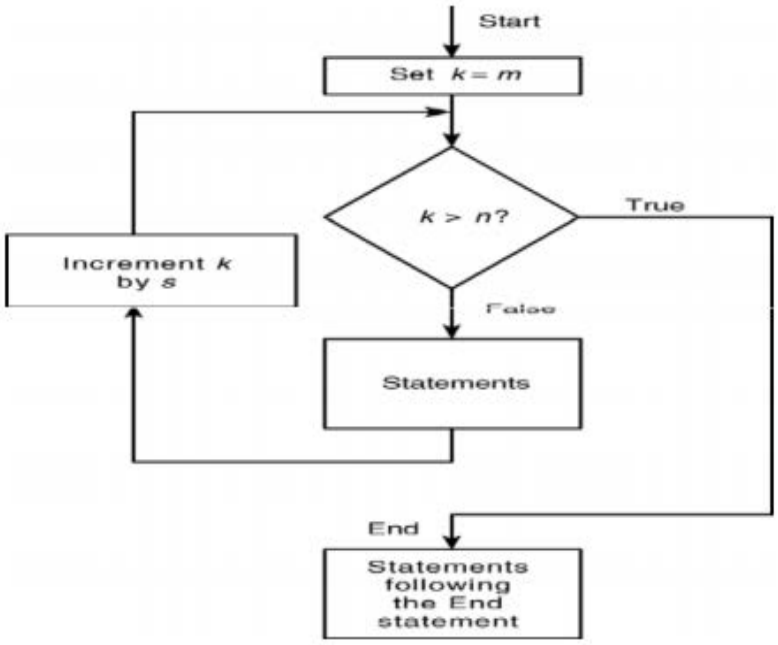
Loops

For Loops:

- Used to repeat a set of statements for specific number of times.



- The loop variable k is initially assigned the value 5, and x is calculated from x = k^2. Each successive pass through the loop increments k by 10 and calculates x until k exceeds 35. Thus, k takes on the values 5, 15, 25, and 35, and x takes on the values 25, 225, 625, and 1225. The program then continues to execute any statements following the end statement.



- Note the following rules when using for loops with the loop variable expression k = [m:s:n]:
  1. The step value s may be negative. Example: k = 10:-2:4 produces k = 10, 8, 6, 4.
  2. If s is omitted, the step value defaults to 1.
  3. If s is positive, the loop will not be executed if m is greater than n.
  4. If s is negative, the loop will not be executed if m is less than n.
  5. If m equals n, the loop will be executed only once.
  6. If the step value s is not an integer, round-off errors can cause the loop to execute a different number of passes than intended.

While Loops:

- The while loop is used when the looping process terminates because a specified condition is satisfied, and thus the number of passes is not known in advance.
- The structure of a while loop:

```
while logical expression
      statements
end
```

- For the while loop to function properly, the following two conditions must occur:

1. The loop variable must have a value before the while statement is executed.
2. The loop variable must be changed somehow by the statements.

- A simple example of a while loop is:

```
x = 5;
while x < 25
    disp(x)
    x = 2*x-1;
end
```

The results displayed by the disp statement are 5, 9, and 17.

- You can create an infinite while loop which is a loop that never ends:

```
x = 8;
while x ~= 0;
    x = x-3;
end
```

- Within the loop the variable x takes on the values 5, 2, -1, -4….., and the condition x~=0 is always satisfied, so the loop never stops.

Loop Vectorization

- We can often avoid the use of loops and branching and thus create simpler and faster programs by using a logical array as a mask that selects elements of another array. This called loop vectorization.
- Here is one way to compute the sine of 1001 values ranging from 0 to 10:

```
i = 0;
for t = [0:0.01:10]
     i = i+1;
     y(i) = sin(t);
end
```

- A vectorized version of the same code is:

```
t = [0:0.01:10];
y = sin(t);
```

- The second example executes much faster than the first and is the way Matlab is meant to be used.
- Example:
  For the array A = [0, -1, 4; 9, -14, 25; -34, 49, 64], find all the elements that are greater than or equal to 0.

  Using Loops:

```
[m,n] = size(A) ;
C = zeros(m,n) ;
for k = 1:m*n
if A(k) >= 0
C(k) = A(k)
end
end
```

  Using array masking and relational operators to vectorize the loop:

```
C = A ;
C(C<0) = 0 ;
```

The Switch Structure

The switch structure provides an alternative to using the if, elseif, and else commands. Anything programmed using switch can also be programmed using if structures. However, for some applications the switch structure is more readable than code using the if structure.

```
switch input expression
    case value1
        statement group 1
    case value2
        statement group 2
    .
    .
    .
    otherwise
        statement group n
    end
```

The following switch block displays the point on the compass that corresponds to that angle.

```
>> switch angle
case 45
disp('Northeast')
case 135
disp('Southeast')
case 225
disp('Southwest')
case 315
disp('Northwest')
otherwise
disp('Direction Unknown')
end
```

Program Design and Development – Finding Bugs

Debugging a program is the process of finding and removing the "bugs," or errors, in a program. Such errors usually fall into one of the following categories.

1. Syntax errors such as omitting a parenthesis or comma, or spelling a command name incorrectly. Matlab usually detects the more obvious errors and displays a message describing the error and its location.
2. Runtime errors; These errors occur due to an incorrect mathematical procedure. They do not necessarily occur every time the program is executed; their occurrence often depends on particular input data. A common example is division by zero.

How to locate your errors?

1. Always test your program with a simple version of the problem, whose answers can be checked by hand calculations.
2. Display any intermediate calculations by removing semicolons at the end of statements.
3. To test user-defined functions, try commenting out the function line and running the file as a script.