CHAPTER 2

2.1 Two possible versions can be developed:

```
IF x ≥ 10 THEN
  DO
    x = x – 5
    IF x < 50 EXIT
  END DO
ELSE
  IF x < 5 THEN
    x = 5
  ELSE
    x = 7.5
  END IF
ENDIF
```

```
IF x ≥ 10 THEN
  DO
    x = x – 5
    IF x < 50 EXIT
  END DO
ELSEIF x < 5
  x = 5
ELSE
  x = 7.5
ENDIF
```

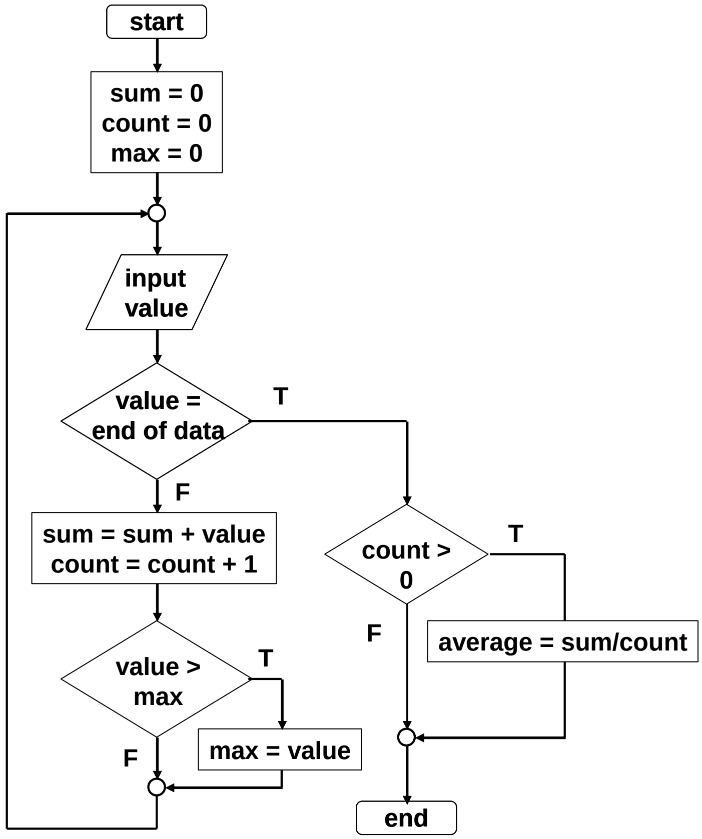
2.2

```
DO
  i = i + 1
  IF z > 50 EXIT
  x = x + 5
  IF x > 5 THEN
    y = x
  ELSE
    y = 0
  ENDIF
  z = x + y
ENDDO
```

2.3 Note that this algorithm is made simpler by recognizing that concentration cannot by definition be negative. Therefore, the maximum can be initialized as zero at the start of the algorithm.

Step 1: Start
Step 2: Initialize ***sum***, ***count*** and **maximum** to zero
Step 3: Examine top card.
Step 4: If it says "end of data" proceed to step 9; otherwise, proceed to next step.
Step 5: Add ***value*** from top card to ***sum***.
Step 6: Increase ***count*** by 1.
Step 7: If ***value*** is greater than ***maximum***, set ***maximum*** to ***value***.
Step 7: Discard top card
Step 8: Return to Step 3.
Step 9: Is the ***count*** greater than zero?
If yes, proceed to step 10.
If no, proceed to step 11.
Step 10: Calculate ***average*** = ***sum***/***count***
Step 11: End

2.4 Flowchart:



2.5 Students could implement the subprogram in any number of languages. The following Fortran 90 program is one example. It should be noted that the availability of complex variables in Fortran 90 would allow this subroutine to be made even more concise. However, we did not exploit this feature, in order to make the code more compatible with languages such as Visual BASIC or C.

```
PROGRAM Rootfind
IMPLICIT NONE
INTEGER::ier
REAL::a, b, c, r1, i1, r2, i2
DATA a,b,c/1.,6.,2./
CALL Roots(a, b, c, ier, r1, i1, r2, i2)
IF (ier == 0) THEN
  PRINT *, r1,i1," i"
  PRINT *, r2,i2," i"
ELSE
  PRINT *, "No roots"
END IF
END

SUBROUTINE Roots(a, b, c, ier, r1, i1, r2, i2)
IMPLICIT NONE
INTEGER::ier
REAL::a, b, c, d, r1, i1, r2, i2
r1=0.
r2=0.
```

```
i1=0.
i2=0.
IF (a == 0.) THEN
  IF (b /= 0) THEN
    r1 = -c/b
  ELSE
    ier = 1
  END IF
ELSE
  d = b**2 - 4.*a*c
  IF (d >= 0) THEN
    r1 = (-b + SQRT(d))/(2*a)
    r2 = (-b - SQRT(d))/(2*a)
  ELSE
    r1 = -b/(2*a)
    r2 = r1
    i1 = SQRT(ABS(d))/(2*a)
    i2 = -i1
  END IF
END IF
END
```

The answers for the 3 test cases are: (*a*) –0.3542, –5.646; (*b*) 0.4; (*c*) –0.4167 + 1.4696*i*; –0.4167 – 1.4696*i*.

Several features of this subroutine bear mention:

- The subroutine does not involve input or output. Rather, information is passed in and out via the arguments. This is often the preferred style, because the I/O is left to the discretion of the programmer within the calling program.
- Note that an error code is passed (`IER = 1`) for the case where no roots are possible.

2.6 The development of the algorithm hinges on recognizing that the series approximation of the cosine can be represented concisely by the summation,

$$\sum_{i=1}^{n} (-1)^{i-1} \frac{x^{2i-2}}{(2i-2)!}$$

where i = the order of the approximation. The following algorithm implements this summation:

Step 1: Start
Step 2: Input value to be evaluated x and maximum order n
Step 3: Set order (i) equal to one
Step 4: Set accumulator for approximation (approx) to zero
Step 5: Set accumulator for factorial product (factor) equal to one
Step 6: Calculate true value of cos(x)
Step 7: If order is greater than n then proceed to step 13
Otherwise, proceed to next step
Step 8: Calculate the approximation with the formula

$$\text{approx} = \text{approx} + (-1)^{i-1}\frac{x^{2i-2}}{\text{factor}}$$

Step 9: Determine the error

$$\%\text{error} = \left|\frac{\text{true} - \text{approx}}{\text{true}}\right| 100\%$$

Step 10: Increment the order by one

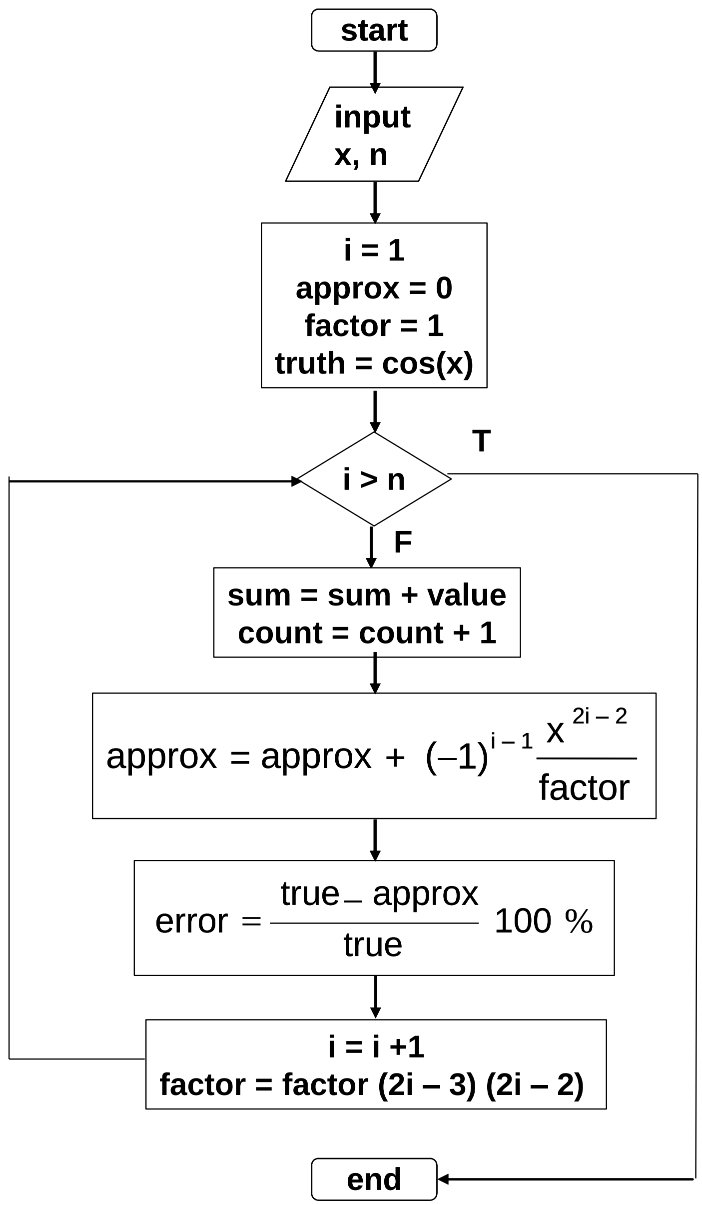
Step 11: Determine the factorial for the next iteration

$$\text{factor} = \text{factor} \bullet (2 \bullet i - 3) \bullet (2 \bullet i - 2)$$

Step 12: Return to step 7

Step 13: End

2.7 (a) Structured flowchart



(b) Pseudocode:

```
SUBROUTINE Coscomp(n,x)
i = 1
approx = 0
factor = 1
truth = cos(x)
DO
  IF i > n EXIT
  approx = approx + (-1)^(i-1)•x^(2•i-2) / factor
  error = (true - approx) / true) * 100
  DISPLAY i, true, approx, error
  i = i + 1
  factor = factor•(2•i-3)•(2•i-2)
END DO
END
```

2.8 Students could implement the subprogram in any number of languages. The following MATLAB M-file is one example. It should be noted that MATLAB allows direct calculation of the factorial through its intrinsic function `factorial`. However, we did not exploit this feature, in order to make the code more compatible with languages such as Visual BASIC and Fortran.

```
function coscomp(x,n)
i = 1;
tru = cos(x);
approx = 0;
f = 1;
fprintf('\n');
fprintf('order  true value      approximation      error\n');
while (1)
  if i > n, break, end
  approx = approx + (-1)^(i - 1) * x^(2*i-2) / f;
  er = (tru - approx) / tru * 100;
  fprintf('%3d  %14.10f  %14.10f  %12.8f\n',i,tru,approx,er);
  i = i + 1;
  f = f*(2*i-3)*(2*i-2);
end
```

Here is a run of the program showing the output that is generated:

```
>> coscomp(1.25,6)

order  true value     approximation     error
  1    0.3153223624    1.0000000000  -217.13576938
  2    0.3153223624    0.2187500000   30.62655045
  3    0.3153223624    0.3204752604   -1.63416828
  4    0.3153223624    0.3151770698    0.04607749
  5    0.3153223624    0.3153248988   -0.00080437
  6    0.3153223624    0.3153223323    0.00000955
```

2.9 (a) The following pseudocode provides an algorithm for this problem. Notice that the input of the quizzes and homeworks is done with logical loops that terminate when the user enters a negative grade:

```
INPUT WQ, WH, WF
nq = 0
sumq = 0
DO
  INPUT quiz (enter negative to signal end of quizzes)
  IF quiz < 0 EXIT
  nq = nq + 1
  sumq = sumq + quiz
END DO
AQ = sumq / nq
nh = 0
sumh = 0
DO
  INPUT homework (enter negative to signal end of homeworks)
  IF homework < 0 EXIT
  nh = nh + 1
  sumh = sumh + homework
END DO
AH = sumh / nh
DISPLAY "Is there a final grade (y or n)"
INPUT answer
IF answer = "y" THEN
  INPUT FE
  AG = (WQ * AQ + WH * AH + WF * FE) / (WQ + WH + WF)
ELSE
  AG = (WQ * AQ + WH * AH) / (WQ + WH)
END IF
DISPLAY AG
END
```

(b) Students could implement the program in any number of languages. The following VBA code is one example.

```
Sub Grader()
Dim WQ As Double, WH As Double, WF As Double
Dim nq As Integer, sumq As Double, AQ As Double
Dim nh As Integer, sumh As Double, AH As Double
Dim answer As String, FE As Double
Dim AG As Double

'enter weights
WQ = InputBox("enter quiz weight")
WH = InputBox("enter homework weight")
WF = InputBox("enter final exam weight")
'enter quiz grades
nq = 0
sumq = 0
Do
  quiz = InputBox("enter negative to signal end of quizzes")
  If quiz < 0 Then Exit Do
```

```
    nq = nq + 1
    sumq = sumq + quiz
  Loop
  AQ = sumq / nq
  'enter homework grades
  nh = 0
  sumh = 0
  Do
    homework = InputBox("enter negative to signal end of homeworks")
    If homework < 0 Then Exit Do
    nh = nh + 1
    sumh = sumh + homework
  Loop
  AH = sumh / nh
  'determine and display the average grade
  answer = InputBox("Is there a final grade (y or n)")
  If answer = "y" Then
    FE = InputBox("final grade:")
    AG = (WQ * AQ + WH * AH + WF * FE) / (WQ + WH + WF)
  Else
    AG = (WQ * AQ + WH * AH) / (WQ + WH)
  End If
  MsgBox "Average grade = " & AG
  End Sub
```

The results should conform to:

AQ = 437/5 = 87.4
AH = 541/6 = 90.1667

without final

$$\mathrm{AG} = \frac{35(87.4) + 30(90.1667)}{35 + 30} = 88.677$$

with final

$$\mathrm{AG} = \frac{35(87.4) + 30(90.1667) + 35(92)}{35 + 30 + 35} = 89.84$$

2.10 (a) Pseudocode:

```
IF a > 0 THEN
  tol = 10^-5
  x = a/2
  DO
    y = (x + a/x)/2
    e = |(y - x)/y|
    x = y
    IF e < tol EXIT
  END DO
  SquareRoot = x
ELSE
  SquareRoot = 0
END IF
```

(b) Students could implement the function in any number of languages. The following VBA and MATLAB codes are two possible options.

<table>
<tr><th>VBA Function Procedure</th><th>MATLAB M-File</th></tr>
<tr><td><pre>Option Explicit
Function SquareRoot(a)
Dim x As Double, y As Double
Dim e As Double, tol As Double
If a > 0 Then
  tol = 0.00001
  x = a / 2
  Do
    y = (x + a / x) / 2
    e = Abs((y - x) / y)
    x = y
    If e < tol Then Exit Do
  Loop
  SquareRoot = x
Else
  SquareRoot = 0
End If
End Function</pre></td><td><pre>function s = SquareRoot(a)
if a > 0
  tol = 0.00001;
  x = a / 2;
  while(1)
    y = (x + a / x) / 2;
    e = abs((y - x) / y);
    x = y;
    if e < tol, break, end
  end
  s = x;
else
  s = 0;
end</pre></td></tr>
</table>

2.11 A MATLAB M-file can be written to solve this problem as

```
function futureworth(P, i, n)
nn = 0:n;
F = P*(1+i).^nn;
y = [nn;F];
fprintf('\n  year   future worth\n');
fprintf('%5d %14.2f\n',y);
```

This function can be used to evaluate the test case,

```
>> futureworth(100000,0.06,5)

  year   future worth
    0      100000.00
    1      106000.00
    2      112360.00
    3      119101.60
    4      126247.70
    5      133822.56
```

2.12 A MATLAB M-file can be written to solve this problem as

```
function annualpayment(P, i, n)
nn = 1:n;
A = P*i*(1+i).^nn./((1+i).^nn-1);
y = [nn;A];
fprintf('\n  year   annual payment\n');
fprintf('%5d %14.2f\n',y);
```

This function can be used to evaluate the test case,

```
>> annualpayment(55000,0.066,5)

  year   annual payment
    1       58630.00
    2       30251.49
    3       20804.86
    4       16091.17
    5       13270.64
```

2.13 Students could implement the function in any number of languages. The following VBA and MATLAB codes are two possible options.

VBA Function Procedure	MATLAB M-File

`Option Explicit`
`Function avgtemp(Tm, Tp, ts, te)`
`Dim pi As Double, w As Double`
`Dim Temp As Double, t As Double`
`Dim sum As Double, i As Integer`
`Dim n As Integer`
`pi = 4 * Atn(1)`
`w = 2 * pi / 365`
`sum = 0`
`n = 0`
`t = ts`
`For i = ts To te`
`  Temp = Tm+(Tp-Tm)*Cos(w*(t-205))`
`  sum = sum + Temp`
`  n = n + 1`
`  t = t + 1`
`Next i`
`avgtemp = sum / n`
`End Function`
	`function Ta = avgtemp(Tm,Tp,ts,te)`
	`w = 2*pi/365;`
	`t = ts:te;`
	`T = Tm + (Tp-Tm)*cos(w*(t-205));`
	`Ta = mean(T);`

The function can be used to evaluate the test cases. The following show the results for MATLAB,

```
>> avgtemp(22.1,28.3,0,59)

ans =
   16.2148

>> avgtemp(10.7,22.9,180,242)

ans =
   22.2491
```

2.14 The programs are student specific and will be similar to the codes developed for VBA, MATLAB and Fortran as outlined in sections 2.4, 2.5 and 2.6. The numerical results for the different time steps are tabulated below along with an estimate of the absolute value of the true relative error at t = 12 s:

Step	$v(12)$	$\lvert\varepsilon_t\rvert$ (%)
2	49.96	5.2
1	48.70	2.6
0.5	48.09	1.3

The general conclusion is that the error is halved when the step size is halved.

2.15 Students could implement the subprogram in any number of languages. The following Fortran 90 and VBA/Excel programs are two examples based on the algorithm outlined in Fig. P2.15.

Fortran 90

```
Subroutine BubbleFor(n, b)

Implicit None

!sorts an array in ascending
!order using the bubble sort

Integer(4)::m, i, n
Logical::switch
Real::a(n),b(n),dum

m = n - 1
Do
  switch = .False.
  Do i = 1, m
    If (b(i) > b(i + 1)) Then
      dum = b(i)
      b(i) = b(i + 1)
      b(i + 1) = dum
      switch = .True.
    End If
  End Do
  If (switch == .False.) Exit
  m = m - 1
End Do

End
```

VBA/Excel

```
Option Explicit

Sub Bubble(n, b)

'sorts an array in ascending
'order using the bubble sort

Dim m As Integer
Dim i As Integer
Dim switch As Boolean
Dim dum As Double

m = n - 1
Do
  switch = False
  For i = 1 To m
    If b(i) > b(i + 1) Then
      dum = b(i)
      b(i) = b(i + 1)
      b(i + 1) = dum
      switch = True
    End If
  Next i
  If switch = False Then Exit Do
  m = m - 1
Loop

End Sub
```

For MATLAB, the following M-file implements the bubble sort following the algorithm outlined in Fig. P2.15:

```
function y = Bubble(x)
n = length(x);
m = n - 1;
b = x;
while(1)
  s = 0;
  for i = 1:m
    if b(i) > b(i + 1)
      dum = b(i);
```

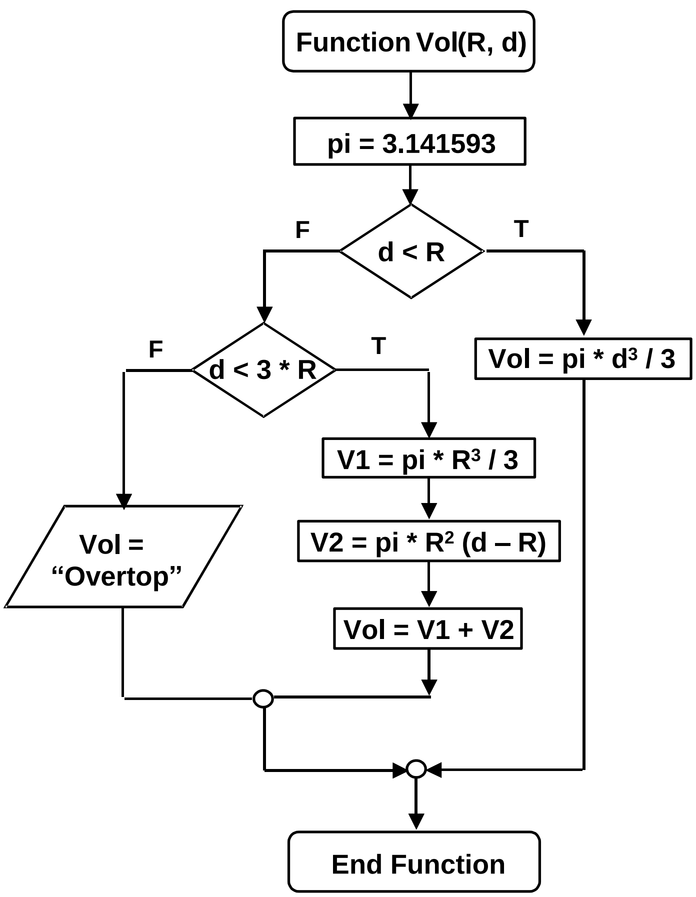
```
      b(i) = b(i + 1);
      b(i + 1) = dum;
      s = 1;
    end
  end
  if s == 0, break, end
  m = m - 1;
end
y = b;
```

Notice how the `length` function allows us to omit the length of the vector in the function argument. Here is an example MATLAB session that invokes the function to sort a vector:

```
>> a=[3 4 2 8 5 7];
>> Bubble(a)

ans =
     2     3     4     5     7     8
```

2.16 Here is a flowchart for the algorithm:



Students could implement the function in any number of languages. The following VBA and MATLAB codes are two possible options.

VBA Function Procedure

```
Option Explicit
Function Vol(R, d)
Dim V1 As Double, V2 As Double
Dim pi As Double
pi = 4 * Atn(1)
If d < R Then
  Vol = pi * d ^ 3 / 3
ElseIf d <= 3 * R Then
  V1 = pi * R ^ 3 / 3
  V2 = pi * R ^ 2 * (d - R)
  Vol = V1 + V2
Else
  Vol = "overtop"
End If
End Function
```

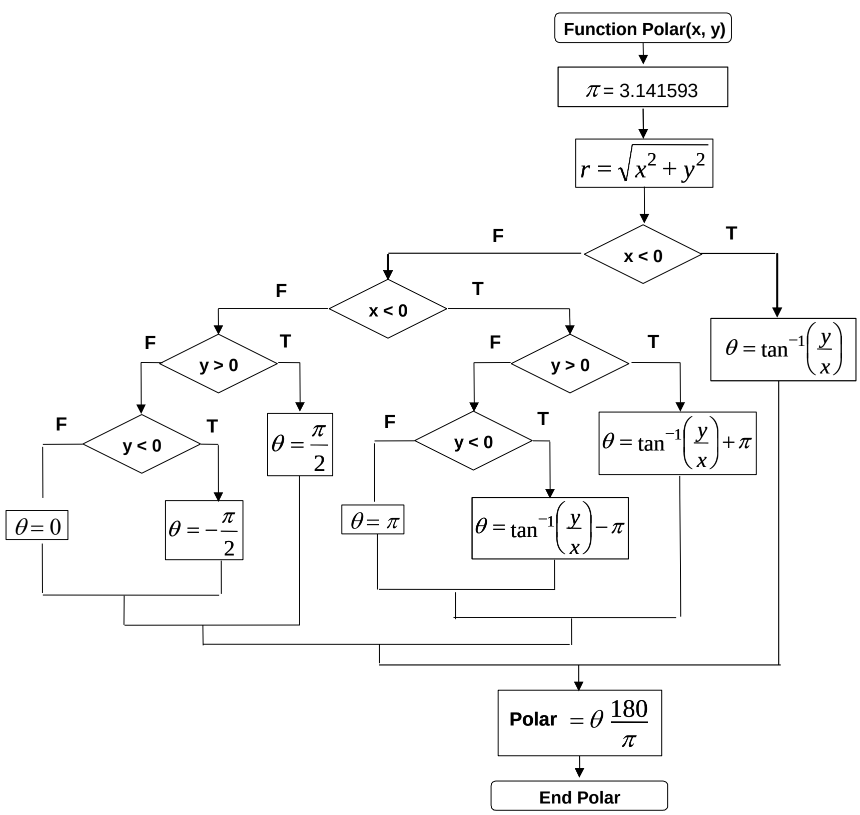
MATLAB M-File

```
function Vol = tankvolume(R, d)
if d < R
  Vol = pi * d ^ 3 / 3;
elseif d <= 3 * R
  V1 = pi * R ^ 3 / 3;
  V2 = pi * R ^ 2 * (d - R);
  Vol = V1 + V2;
else
  Vol = 'overtop';
end
```

The results are:

R	d	Volume
1	0.5	0.1309
1	1.2	1.675516
1	3	7.330383
1	3.1	overtop

2.17 Here is a flowchart for the algorithm:



Students could implement the function in any number of languages. The following MATLAB M-file is one option. Versions in other languages such as Fortran 90, Visual Basic, or C would have a similar structure.

```
function polar(x, y)
r = sqrt(x .^ 2 + y .^ 2);
n = length(x);
for i = 1:n
  if x(i) > 0
    th(i) = atan(y(i) / x(i));
  elseif x(i) < 0
    if y(i) > 0
      th(i) = atan(y(i) / x(i)) + pi;
    elseif y(i) < 0
      th(i) = atan(y(i) / x(i)) - pi;
    else
      th(i) = pi;
    end
  else
    if y(i) > 0
      th(i) = pi / 2;
    elseif y(i) < 0
      th(i) = -pi / 2;
    else
      th(i) = 0;
    end
  end
  th(i) = th(i) * 180 / pi;
end
ou=[x;y;r;th];
fprintf('\n     x        y       radius     angle\n');
fprintf('%8.2f %8.2f %10.4f %10.4f\n',ou);
```

This function can be used to evaluate the test cases.

```
>> x=[1 1 1 -1 -1 -1 0 0 0];
>> y=[1 -1 0 1 -1 0 1 -1 0];
>> polar(x,y)

     x        y       radius     angle
    1.00     1.00     1.4142    45.0000
    1.00    -1.00     1.4142   -45.0000
    1.00     0.00     1.0000     0.0000
   -1.00     1.00     1.4142   135.0000
   -1.00    -1.00     1.4142  -135.0000
   -1.00     0.00     1.0000   180.0000
    0.00     1.00     1.0000    90.0000
    0.00    -1.00     1.0000   -90.0000
    0.00     0.00     0.0000     0.0000
```

2.18 Students could implement the function in any number of languages. The following VBA and MATLAB codes are two possible options.

VBA Function Procedure	MATLAB M-File

`Function grade(s)`
`If s >= 90 Then`
`  grade = "A"`
`ElseIf s >= 80 Then`
`  grade = "B"`
`ElseIf s >= 70 Then`
`  grade = "C"`
`ElseIf s >= 60 Then`
`  grade = "D"`
`Else`
`  grade = "F"`
`End If`
`End Function`
	`function grade = lettergrade(score)`
	`if score >= 90`
	`  grade = 'A';`
	`elseif score >= 80`
	`  grade = 'B';`
	`elseif score >= 70`
	`  grade = 'C';`
	`elseif score >= 60`
	`  grade = 'D';`
	`else`
	`  grade = 'F';`
	`end`

2.19 Students could implement the functions in any number of languages. The following VBA and MATLAB codes are two possible options.

VBA Function Procedure	MATLAB M-File

(a) Factorial
`Function factor(n)`
`Dim x As Long, i As Integer`
`x = 1`
`For i = 1 To n`
`  x = x * i`
`Next i`
`factor = x`
`End Function`
	
	`function fout = factor(n)`
	`x = 1;`
	`for i = 1:n`
	`  x = x * i;`
	`end`
	`fout = x;`

(b) Minimum
`Function min(x, n)`
`Dim i As Integer`
`min = x(1)`
`For i = 2 To n`
`  If x(i) < min Then min = x(i)`
`Next i`
`End Function`
	
	`function xm = xmin(x)`
	`n = length(x);`
	`xm = x(1);`
	`for i = 2:n`
	`  if x(i) < xm, xm = x(i); end`
	`end`

(c) Average
`Function mean(x, n)`
`Dim sum As Double`
`Dim i As Integer`
`sum = x(1)`
`For i = 2 To n`
`  sum = sum + x(i)`
`Next i`
`mean = sum / n`
`End Function`
	
	`function xm = xmean(x)`
	`n = length(x);`
	`s = x(1);`
	`for i = 2:n`
	`  s = s + x(i);`
	`end`
	`xm = s / n;`

2.20 Students could implement the functions in any number of languages. The following VBA and MATLAB codes are two possible options.

VBA Function Procedure	MATLAB M-File
(a) Square root sum of squares	

```
Function SSS(x, n, m)
Dim i As Integer, j As Integer
SSS = 0
For i = 1 To n
  For j = 1 To m
    SSS = SSS + x(i, j) ^ 2
  Next j
Next i
SSS = Sqr(SSS)
End Function
```

```
function s = SSS(x)
[n,m] = size(x);
s = 0;
for i = 1:n
  for j = 1:m
    s = s + x(i, j) ^ 2;
  end
end
s = sqrt(s);
```

(b) Normalization

```
Sub normal(x, n, m, y)
Dim i As Integer, j As Integer
Dim max As Double
For i = 1 To n
  max = Abs(x(i, 1))
  For j = 2 To m
    If Abs(x(i, j)) > max Then
      max = x(i, j)
    End If
  Next j
  For j = 1 To m
    y(i, j) = x(i, j) / max
  Next j
Next i
End Sub
```

```
function y = normal(x)
[n,m] = size(x);
for i = 1:n
  mx = abs(x(i, 1));
  for j = 2:m
    if abs(x(i, j)) > mx
      mx = x(i, j);
    end
  end
  for j = 1:m
    y(i, j) = x(i, j) / mx;
  end
end

Alternate version:

function y = normal(x)
n = size(x);
for i = 1:n
  y(i,:) = x(i,:)/max(x(i,:));
end
```