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 Students could implement the subprogram in any number of languages. The following VBA program is one example. It should be noted that the availability of complex variables in languages such as 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 that do not support complex variables.

```
Option Explicit

Sub Rootfind()
Dim ier As Integer
Dim a As Double, b As Double, c As Double
Dim r1 As Double, i1 As Double, r2 As Double, i2 As Double
a = 7: b = 6: c = 2
Call Roots(a, b, c, ier, r1, i1, r2, i2)
If ier = 0 Then
  MsgBox "No roots"
ElseIf ier = 1 Then
  MsgBox "single root=" & r1
ElseIf ier = 2 Then
  MsgBox "real roots = " & r1 & ", " & r2
ElseIf ier = 3 Then
  MsgBox "complex roots =" & r1 & "," & i1 & " i" & "; "_
                         & r2 & "," & i2 & " i"
End If
End Sub

Sub Roots(a, b, c, ier, r1, i1, r2, i2)
Dim d As Double
r1 = 0: r2 = 0: i1 = 0: i2 = 0
```

```
If a = 0 Then
  If b <> 0 Then
    r1 = -c / b
    ier = 1
  Else
    ier = 0
  End If
Else
  d = b ^ 2 - 4 * a * c
  If (d >= 0) Then
    r1 = (-b + Sqr(d)) / (2 * a)
    r2 = (-b - Sqr(d)) / (2 * a)
    ier = 2
  Else
    r1 = -b / (2 * a)
    r2 = r1
    i1 = Sqr(Abs(d)) / (2 * a)
    i2 = -i1
    ier = 3
  End If
End If
End Sub
```

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:

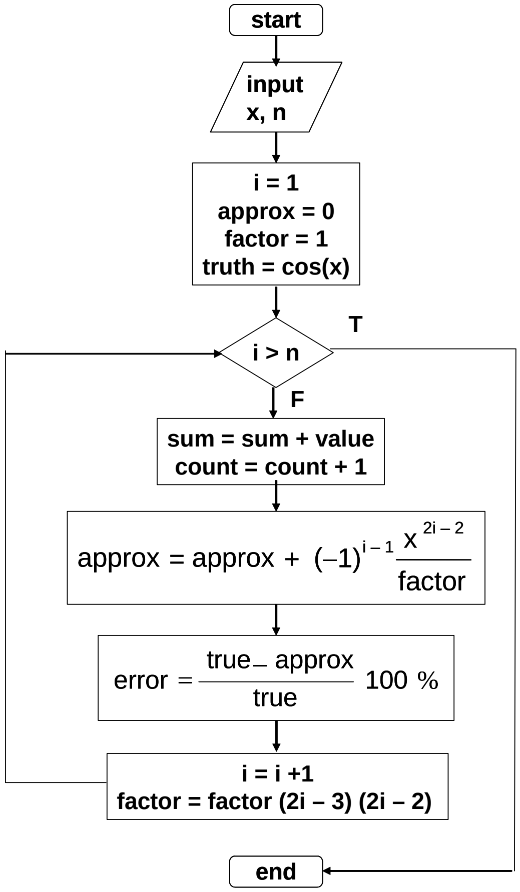
- The subroutine does not involve input or output. Rather, information is passed in and out via the arguments. This is often the preferred style, because the I/O is left to the discretion of the programmer within the calling program.
- Note that a variable is passed (`IER`) in order to distinguish among the various cases.

2.4 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.

(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
```

2.5 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
```

2.6 **(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

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

with final

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

2.7 (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.

VBA Function Procedure	MATLAB M-File

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
	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

2.8 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
```

2.9 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
```

2.10 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
```

2.11 The programs are student specific and will be similar to the codes developed for VBA and MATLAB as outlined in sections 2.4 and 2.5. 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.

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

VBA/Excel

```
Option Explicit

Sub Bubble(n, b)
Dim m As Integer, i As Integer
Dim switch As Boolean, 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
```

MATLAB

```
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 MATLAB `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
```

2.13 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
```

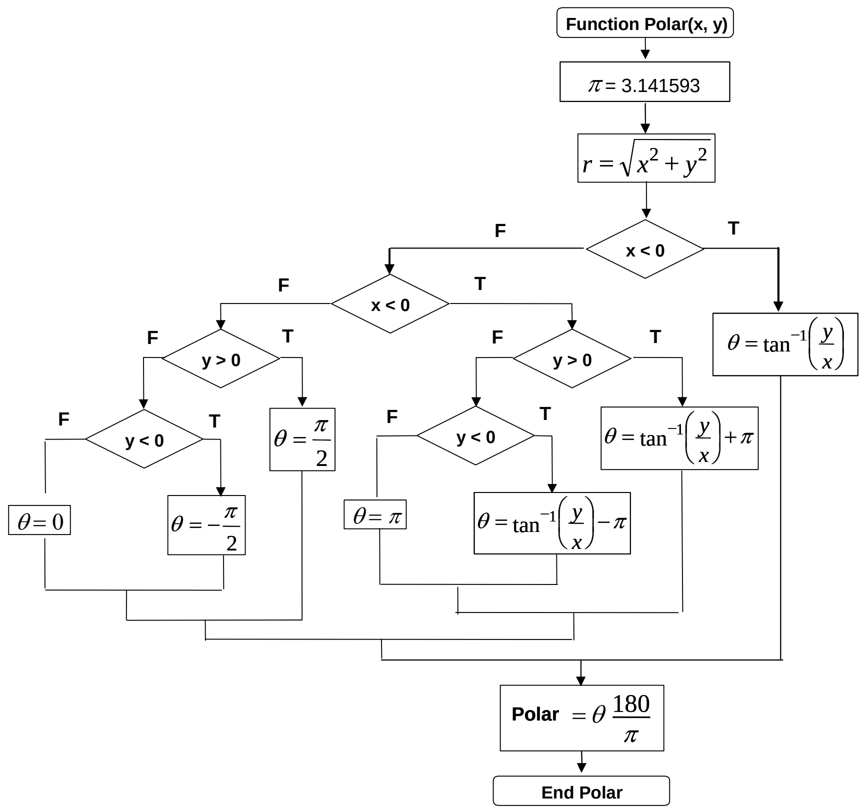
MATLAB M-File

```
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

2.14 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
```

2.15 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`

2.16 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`
	
	`function fout = factor(n)`
	`x = 1;`
	`for i = 1:n`
	`  x = x * i;`
	`end`
	`fout = x;`

```
End Function
```

(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;
```

2.17 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

VBA Function Procedure:

```
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
```

MATLAB M-File:

```
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

VBA Function Procedure:

```
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
```

MATLAB M-File:

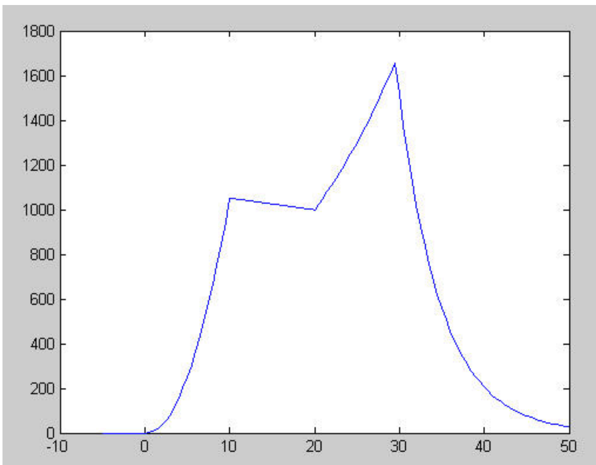
```
function y = normal(x)
[n,m] = size(x);
for i = 1:n
  mx = abs(x(i, 1));
  for j = 2:m
    if abs(x(i, j)) > mx
      mx = x(i, j);
    end
  end
  for j = 1:m
    y(i, j) = x(i, j) / mx;
  end
end

Alternate version:

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

2.18 The following MATLAB function implements the piecewise function:

```
function v = vpiece(t)
if t<0
  v = 0;
elseif t<10
  v = 11*t^2 - 5*t;
elseif t<20
  v = 1100 - 5*t;
elseif t<30
  v = 50*t + 2*(t - 20)^2;
else
  v = 1520*exp(-0.2*(t-30));
end
```

Here is a script that uses `vpiece` to generate the plot

```
k=0;
for i = -5:.5:50
  k=k+1;
  t(k)=i;
  v(k)=vpiece(t(k));
end
plot(t,v)
```



2.19 The following MATLAB function implements the algorithm:

```
function nd = days(mo, da, leap)
nd = 0;
for m=1:mo-1
  switch m
    case {1, 3, 5, 7, 8, 10, 12}
      nday = 31;
    case {4, 6, 9, 11}
      nday = 30;
    case 2
      nday = 28+leap;
  end
  nd=nd+nday;
end
nd = nd + da;
```

```
>> days(1,1,0)
ans =
     1
>> days(2,29,1)
ans =
    60
>> days(3,1,0)
ans =
    60
>> days(6,21,0)
ans =
   172
>> days(12,31,1)
ans =
   366
```

2.20 The following MATLAB function implements the algorithm:

```
function nd = days(mo, da, year)
leap = 0;
if year / 4 - fix(year / 4) == 0, leap = 1; end
nd = 0;
for m=1:mo-1
  switch m
    case {1, 3, 5, 7, 8, 10, 12}
      nday = 31;
    case {4, 6, 9, 11}
      nday = 30;
    case 2
      nday = 28+leap;
  end
  nd=nd+nday;
end
nd = nd + da;

>> days(1,1,1999)
ans =
     1
>> days(2,29,2000)
ans =
    60
>> days(3,1,2001)
ans =
    60
>> days(6,21,2002)
ans =
   172
>> days(12,31,2004)
ans =
   366
```

2.21 A MATLAB M-file can be written as

```
function Manning(A)
A(:,5)=sqrt(A(:,2))./A(:,1).*(A(:,3).*A(:,4)./(A(:,3)+2*A(:,4))).^(2/3);
fprintf('\n    n         S          B          H          U\n');
fprintf('%8.3f %8.4f %10.2f %10.2f %10.4f\n',A');
```

This function can be run to create the table,

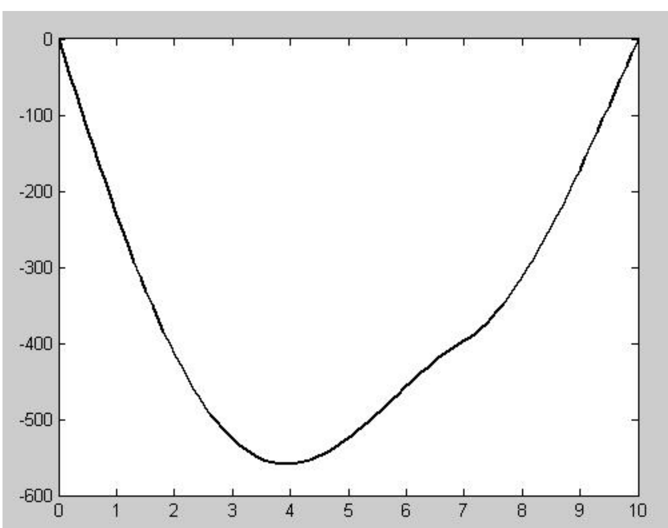
```
>> A=[.035 .0001 10 2
.020 .0002 8 1
.015 .001 20 1.5
.03 .0007 24 3
.022 .0003 15 2.5];
>> Manning(A)

    n          S          B          H          U
  0.035   0.0001      10.00       2.00     0.3624
  0.020   0.0002       8.00       1.00     0.6094
  0.015   0.0010      20.00       1.50     2.5167
  0.030   0.0007      24.00       3.00     1.5809
  0.022   0.0003      15.00       2.50     1.1971
```

2.22 A MATLAB M-file can be written as

```
function beam(x)
xx = linspace(0,x);
n=length(xx);
for i=1:n
  uy(i) = -5/6.*(sing(xx(i),0,4)-sing(xx(i),5,4));
  uy(i) = uy(i) + 15/6.*sing(xx(i),8,3) + 75*sing(xx(i),7,2);
  uy(i) = uy(i) + 57/6.*xx(i)^3 - 238.25.*xx(i);
end
plot(xx,uy)

function s = sing(xxx,a,n)
if xxx > a
  s = (xxx - a).^n;
else
  s=0;
end
```

This function can be run to create the plot,

```
>> beam(10)
```



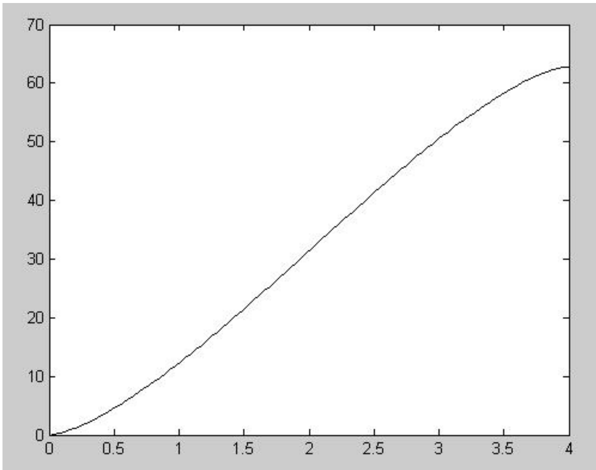
2.23 A MATLAB M-file can be written as

```
function cylinder(r, L)
```

```
h = linspace(0,2*r);
V = (r^2*acos((r-h)./r)-(r-h).*sqrt(2*r*h-h.^2))*L;
plot(h, V)
```

This function can be run to create the plot,

```
>> cylinder(2,5)
```



2.24 Before the chute opens ($t < 10$), Euler's method can be implemented as

$$v(t+\Delta t) = v(t) + \left[9.8 - \frac{10}{80}v(t)\right]\Delta t$$

After the chute opens ($t \geq 10$), the drag coefficient is changed and the implementation becomes

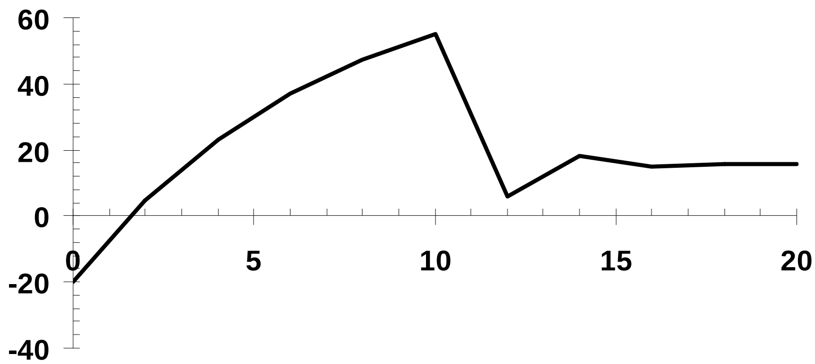
$$v(t+\Delta t) = v(t) + \left[9.8 - \frac{50}{80}v(t)\right]\Delta t$$

You can implement the subprogram in any number of languages. The following MATLAB M-file is one example. Notice that the results are inaccurate because the stepsize is too big. A smaller stepsize should be used to attain adequate accuracy.

```
function parachute
g = 9.81;
m = 80; c = 10;
ti = 0; tf = 20; dt = 2;
vi = -20;
tc = 10; cc = 50;
np = (tf - ti) / dt;
t = ti; v = vi;
tout(1) = t; vout(1) = v;
for i = 1:np
  if t < tc
    dvdt = g - c / m * v;
  else
    dvdt = g - cc / m * v;
  end
  v = v + dvdt * dt;
  t = t + dt;
```

```
  tout(i+1) = t; vout(i+1) = v;
end
plot(tout,vout)
z=[tout;vout]
fprintf('    t        v\n');
fprintf('%5d %10.3f\n',z);
```

```
    t        v
    0    -20.000
    2      4.620
    4     23.085
    6     36.934
    8     47.320
   10     55.110
   12      5.842
   14     18.159
   16     15.080
   18     15.850
   20     15.658
```



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

VBA/Excel	MATLAB

`Option Explicit`
`Function fac(n)`
`Dim x As Long, i As Integer`
`If n >= 0 Then`
`  x = 1`
`  For i = 1 To n`
`    x = x * i`
`  Next i`
`  fac = x`
`Else`
`  MsgBox "value must be positive"`
`  End`
`End If`
`End Function`
	
	`function f = fac(n)`
	
	`if n >= 0`
	`  x = 1;`
	`  for i = 1: n`
	`    x = x * i;`
	`  end`
	`  f = x;`
	`else`
	`  error 'value must be positive'`
	`end`