CHAPTER 2

2.1 Write pseudocode to implement the flowchart depicted in Fig. P2.1. Make sure that proper indentation is included to make the structure clear.

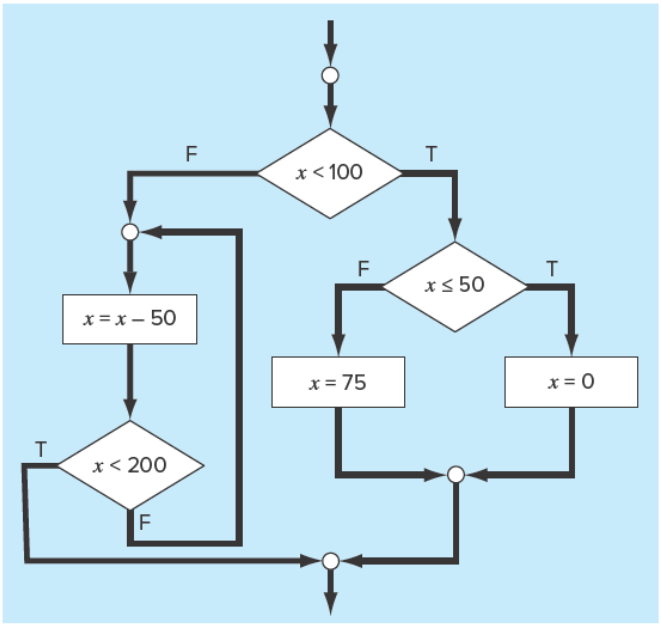


FIGURE P2.1

```
IF x < 100 THEN
  IF x ≤ 50 THEN
    x = 0
  ELSE
    x = 75
  END IF
ELSE
  DO
    x = x - 50
    IF x < 200 EXIT
  END DO
ENDIF
```

2.2 Rewrite the following pseudocode using proper indentation:

```
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
```

```
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 Develop, debug, and document a program to determine the roots of a quadratic equation, $ax^2 + bx + c$, in either a high-level language or a macro language of your choice. Use a subroutine procedure to compute the roots (either real or complex). Perform test runs for the cases **(a)** $a = 1$, $b = 6, c = 2$; **(b)** $a = 0, b = -4, c = 1.6$; **(c)** $a = 3, b = 2.5, c = 7$.

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. This version is then followed by a MATLAB script and function that does accommodate 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 = 1: b = 7: 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*) –0.2984, –6.702; (*b*) 0.32; (*c*) –0.4167 + 1.5789*i*; –0.4167 – 1.5789*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.

MATLAB:

```
function [r1,r2]=quadroots(a,b,c)
r1 = 0; r2 = 0;
if a == 0
  if b ~= 0
    r1=-c/b;
  else
    r1='Trivial solution';
  end
else
  discr=b^2-4*a*c;
  if discr >= 0
    r1=(-b+sqrt(discr))/(2*a);
    r2=(-b-sqrt(discr))/(2*a);
  else
    r1 =-b/(2*a); i1=sqrt(abs(discr))/(2*a);
    r2=r1-i1*i; r1=r1+i1*i;
  end
end
```

Script:

```
clc
format compact
disp('(a)'),[r1,r2]=quadroots(1,7,2)
disp('(b)'),[r1,r2]=quadroots(0,-5,1.6)
disp('(c)'),[r1,r2]=quadroots(3,2.5,8)
```

Output when script is run

```
(a)
r1 =
   -0.2984
r2 =
   -6.7016
(b)
r1 =
    0.3200
r2 =
     0
(c)
r1 =
  -0.4167 + 1.5789i
r2 =
  -0.4167 - 1.5789i
```

2.4 The sine function can be evaluated by the following infinite series:

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots$$

Write an algorithm to implement this formula so that it computes and prints out the values of cos x as each term in the series is added. In other words, compute and print in sequence the values for

$$\cos x = 1$$

$$\cos x = 1 - \frac{x^2}{2!}$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!}$$

up to the order term n of your choosing. For each of the preceding, compute and display the percent relative error as

$$\% \text{ error} = \frac{\text{true} - \text{series approximation}}{\text{true}} 100\%$$

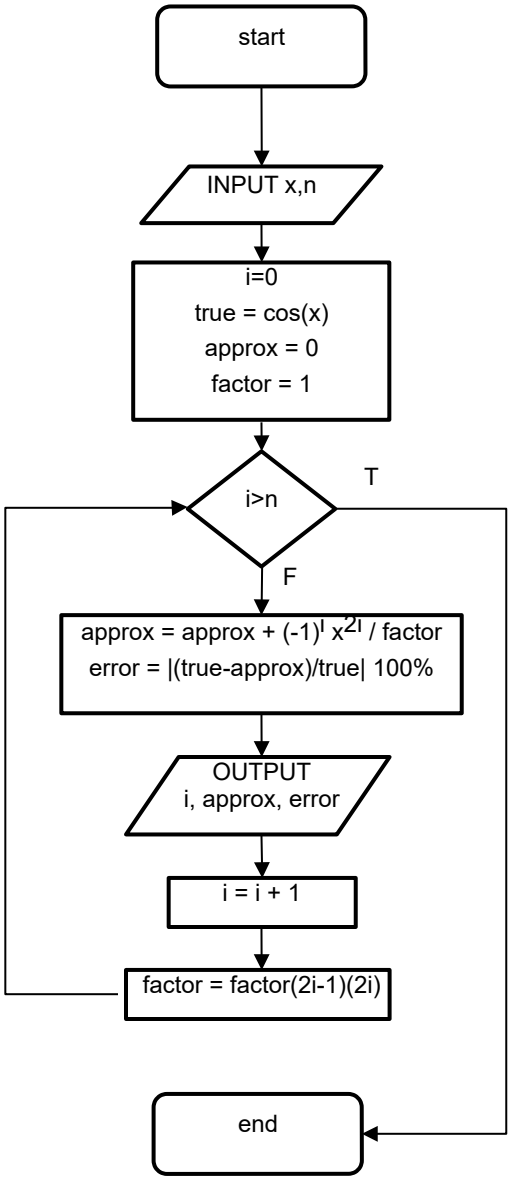
Write the algorithm as **(a)** a structured flowchart and **(b)** pseudocode.

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

$$\sum_{i=0}^{n} (-1)^i \frac{x^{2i}}{(2i)!}$$

where i = the order of the approximation.

(a) Structured flowchart:



(b) Pseudocode:

```
SUBROUTINE Coscomp(n,x)
i = 0; truth = COS(x); approx = 0
factor = 1
DO
  IF i > n EXIT
  approx = approx + (-1)^i·x^2•i / factor
  error = (truth - approx) / truth) * 100
  PRINT i, truth, approx, error
  i = i + 1
  factor = factor·(2·i-1)·(2·i)
END DO
END
```

2.5 Develop, debug, and document a program for Prob. 2.4 in either a high-level language or a macro language of your choice. Employ the library function for the cosine in your computer to determine the true value. Have the program print out the series approximation and the error at each step. As a test case, employ the program to compute sin(1.25) for up to and including the term $x^{10}/10!$. Interpret your results.

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 = 0; 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 * x^(2*i) / f;
  er = abs((tru - approx) / tru )* 100;
  fprintf('%3d  %14.10f  %14.10f  %12.8f \n',i,tru,approx,er);
  i = i + 1;
  f = f*(2*i-1)*(2*i);
end
```

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

```
>> coscomp(1.25,5)

order  true value     approximation     error
  0    0.3153223624    1.0000000000  217.13576938
  1    0.3153223624    0.2187500000   30.62655045
  2    0.3153223624    0.3204752604    1.63416828
  3    0.3153223624    0.3151770698    0.04607749
  4    0.3153223624    0.3153248988    0.00080437
  5    0.3153223624    0.3153223323    0.00000955
```

2.6 The following algorithm is designed to determine a grade for a course that consists of quizzes, homework, and a final exam:

Step 1: Input course number and name.
Step 2: Input weighting factors for quizzes (WQ), homework (WH), and the final exam (WF).
Step 3: Input quiz grades and determine an average quiz grade (AQ).
Step 4: Input homework grades and determine an average homework grade (AH).
Step 5: If this course has a final exam grade, continue to step 6. If not, go to step 9.
Step 6: Input final exam grade (FE).
Step 7: Determine average grade AG according to

$$AG = \frac{WQ' AQ + WH' AH + WF' FE}{WQ + WH + WF}' 100\%$$

Step 8: Go to step 10.
Step 9: Determine average grade AG according to

$$AG = \frac{WQ' AQ + WH' AH}{WQ + WH}' 100\%$$

Step 10: Print out course number, name, and average grade.
Step 11: Terminate computation.
(a) Write well-structured pseudocode to implement this algorithm.

(b) Write, debug, and document a structured computer program based on this algorithm. Test it using the following data to calculate a grade without the final exam and a grade with the final exam: WQ = 35; WH = 30; WF = 35; quizzes = 98, 85, 90, 65, 99; homework = 95, 90, 87, 100, 92, 77; and final exam = 92.

(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 exam (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.

```
Option Explicit

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, quiz As Double, homework 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 exam (y or n)")
If answer = "y" Then
  FE = InputBox("final exam:")
  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

$$\text{AG} = \frac{35(87.4) + 30(90.1667)}{35 + 30} = 88.6769$$

with final

$$\text{AG} = \frac{35(87.4) + 30(90.1667) + 35(92)}{30 + 40 + 30} = 89.8400$$

Here is an example of how a MATLAB script could be developed to solve the same problem:

```
clc
% enter weights
WQ = input('enter quiz weight');
WH = input('enter homework weight');
WF = input('enter final exam weight');
% enter quiz grades
nq = 0; sumq = 0;
while(1)
  quiz = input('enter negative to signal end of quizzes');
  if quiz < 0;break;end
  nq = nq + 1;
  sumq = sumq + quiz;
end
AQ = sumq / nq;
% enter homework grades
nh = 0; sumh = 0;
while(1)
  homework = input('enter negative to signal end of homeworks');
  if homework < 0;break;end
  nh = nh + 1;
  sumh = sumh + homework;
end
AH = sumh / nh;
answer = input('Is there a final exam (y or n)','s');
```

```
if answer == 'y'
  FE = input('final exam:');
  AG = (WQ * AQ + WH * AH + WF * FE) / (WQ + WH + WF);
else
  AG = (WQ * AQ + WH * AH) / (WQ + WH);
end
fprintf('Average grade: %8.4f\n',AG)
```

Finally, here is an alternative MATLAB script that solves the same problem, but is much more concise. Note that rather than using interactive input, the script employs vectors to enter the data. In addition, the nonexistence of a final is denoted by entering a negative number for the final exam:

```
clc
WQ=30;WH=40;WF=30;
QG=[98 95 90 60 99];
HG=[98 95 86 100 100 77];
FE=91;
if FE>0
  AG=(WQ*mean(QG)+WH*mean(HG)+WF*FE)/(WQ+WH+WF);
else
  AG=(WQ*mean(QG)+WH*mean(HG))/(WQ+WH);
end
fprintf('Average grade: %8.4f\n',AG)
```

2.7 The "divide and average" method, an old-time method for approximating the square root of any positive number a, can be formulated as

$$x = \frac{x + a/x}{2}$$

(a) Write well-structured pseudocode to implement this algorithm as depicted in Fig. P2.7. Use proper indentation so that the structure is clear.

(b) Develop, debug, and document a program to implement this equation in either a high-level language or a macro language of your choice. Structure your code according to Fig. P2.7.

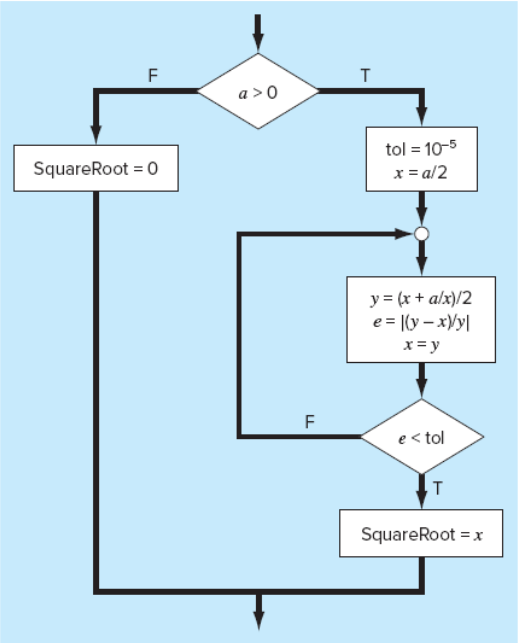


FIGURE P2.7

(a) Pseudocode:

```
IF a > 0 THEN
  tol = 10^-6
  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 = 0.000001`
`  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 = 0.000001;`
	`  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 An amount of money P is invested in an account where interest is compounded at the end of the period. The future worth F yielded at an interest rate i after n periods may be determined from the following formula:

$$F = P(1+i)^n$$

Write a program that will calculate the future worth of an investment for each year from 1 through n. The input to the function should include the initial investment P, the interest rate i (as a decimal), and the number of years n for which the future worth is to be calculated. The output should consist of a table with headings and columns for n and F. Run the program for P = \$100,000, i = 0.06, and n = 5 years.

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

```
function futureworth(P, i, n)
% Calculates future worth of an investment
% input: P principle, i periodic interest rate, n number of periods
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 Economic formulas are available to compute annual payments for loans. Suppose that you borrow an amount of money P and agree to repay it in n annual payments at an interest rate of i. The formula to compute the annual payment A is

$$A = P\frac{i(1+i)^n}{(1+i)^n - 1}$$

Write a program to compute A. Test it with $P = \$55,000$ and an interest rate of 5.6% $(i = 0.056)$. Compute results for $n = 1, 2, 3, 4,$ and 5 and display the results as a table with headings and columns for n and A.

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

```
function annualpayment(P, i, n)
% Calculates the annual / periodic payments on a loan of various lengths
% Inputs:   P amount borrowed, i periodic interest rate, n maximum number of
%           periodic payments
% Outputs:  yearly payment amount for terms of 1 to n years
nn = 1:n;
A = P*i*(1+i).^nn./((1+i).^nn-1);
y = [nn;A];
fprintf('\n  years  annual payment\n');
fprintf('%5d %14.2f\n',y);
```

This function can be used to evaluate the test case,

```
>> annualpayment(55000,0.05,5)

  years  annual payment
    1       57750.00
    2       29579.27
    3       20196.47
    4       15510.65
    5       12703.61
```

2.10 The average daily temperature for an area can be approximated by the following function,

$$T = T_{mean} + (T_{peak} - T_{mean})\cos(\omega(t - t_{peak}))$$

where T_{mean} = the average annual temperature, T_{peak} = the peak temperature, ω = the frequency of the annual variation $(= 2\pi/365)$, and t_{peak} = day of the peak temperature ($\cong$ 205 d). Develop a program that computes the average temperature between two days of the year for a particular city. Test it for **(a)** January–February $(t = 0 \text{ to } 59)$ in Miami, Florida $(T_{mean} = 22.1^\circ\text{C}; T_{peak} = 28.3^\circ\text{C})$, and **(b)** July–August $(t = 180 \text{ to } 242)$ in Boston, Massachusetts $(T_{mean} = 10.7^\circ\text{C}; T_{peak} = 22.9^\circ\text{C})$.

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 Develop, debug, and test a program in either a high-level language or a macro language of your choice to compute the velocity of the falling parachutist as outlined in Example 1.2. Design the program so that it allows the user to input values for the drag coefficient and mass. Test the program by duplicating the results from Example 1.2. Repeat the computation but employ step sizes of 1 and 0.5 s. Compare your results with the analytical solution obtained previously in Example 1.1. Does a smaller step size make the results better or worse? Explain your results.

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. For example, the following MATLAB script was developed to use the function from section 2.5 to compute and tabulate the numerical results for the value at $t = 12$ s, along with an estimate of the absolute value of the true relative error based on the analytical solution:

```
clc; format compact
m=68.1; cd=12.5;
ti=0; tf=12.;
vi=0;
vtrue=9.81*m/cd*(1-exp(-cd/m*tf))
dt=[2 1 0.5]';
for i = 1:3
  v(i)=euler(dt(i),ti,tf,vi,m,cd);
end
et=abs((vtrue-v)/vtrue*100);
z=[dt v' et']';
fprintf('      dt         v(12)      et(pct)\n')
fprintf('%10.3f %10.3f %10.3f\n',z);
```

Output:

```
vtrue =
   47.5387

      dt         v(12)      et(pct)
   2.000      50.010       5.199
   1.000      48.756       2.561
   0.500      48.142       1.269
```

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

2.12 The bubble sort is an inefficient, but easy-to-program, sorting technique. The idea behind the sort is to move down through an array comparing adjacent pairs and swapping the values if they are out of order. For this method to sort the array completely, it may need to pass through it many times. As the passes proceed for an ascending-order sort, the smaller elements in the array appear to rise toward the top like bubbles. Eventually, there will be a pass through the array where no swaps are required. Then, the array is sorted. After the first pass, the largest value in the array drops directly to the bottom. Consequently, the second pass only has to proceed to the second-to-last value, and so on. Develop a program to set up an array of 20 random numbers and sort them in ascending order with the bubble sort (Fig. P2.12).

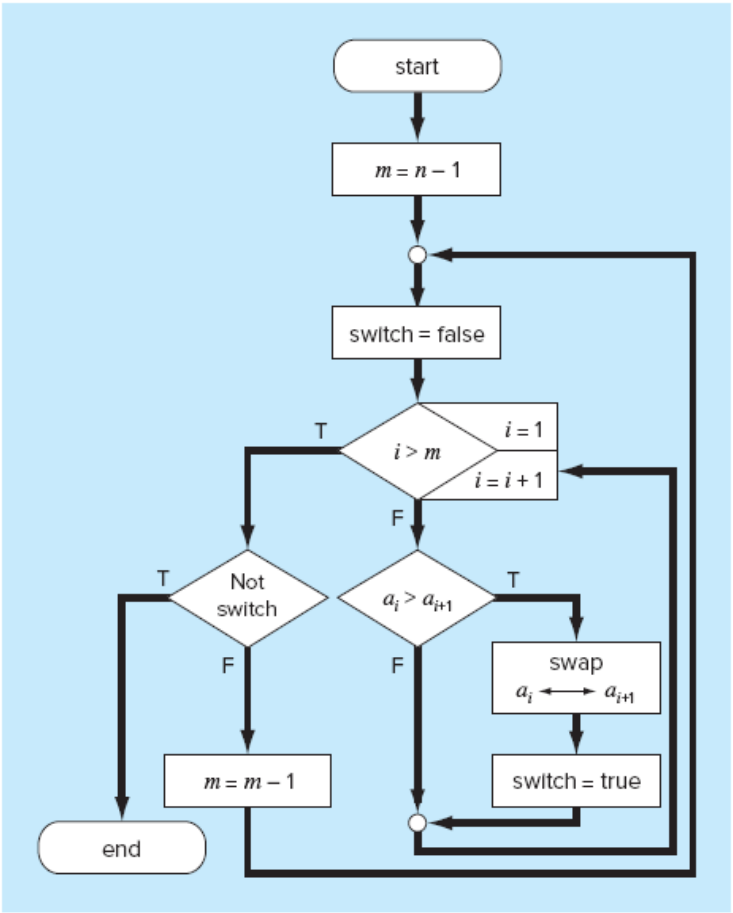


FIGURE P2.12

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. P2.12.

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 script that invokes the function to sort a vector:

```
clc
a=[6 3 4 2 1 5 7];
Bubble(a)

ans =
     1     2     3     4     5     6     7
```

2.13 Figure P2.13 shows a cylindrical tank with a conical base. If the liquid level is quite low in the conical part, the volume is simply the conical volume of liquid. If the liquid level is midrange in the cylindrical part, the total volume of liquid includes the filled conical part and the partially filled cylindrical part. Write a well-structured function procedure to compute the liquid's volume as a function of given values of R and d. Use decisional control structures (like IF/THEN, ELSEIF, ELSE, ENDIF). Design the function so that it returns the volume for all cases where the depth is less than $3R$. Return an error message ("Overtop") if you overtop the tank, that is, $d > 3R$. Test it with the following data:

R	1	1	1	1
d	0.5	1.2	3.0	3.1

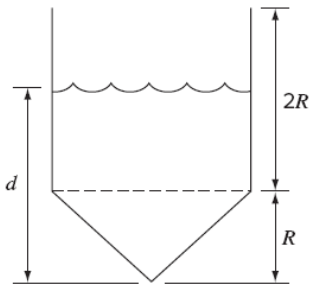


FIGURE P2.13

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 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`
	`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 Two distances are required to specify the location of a point relative to an origin in two-dimensional space (Fig. P2.14):

- The horizontal and vertical distances (x, y) in Cartesian coordinates
- The radius and angle (r, θ) in radial coordinates.

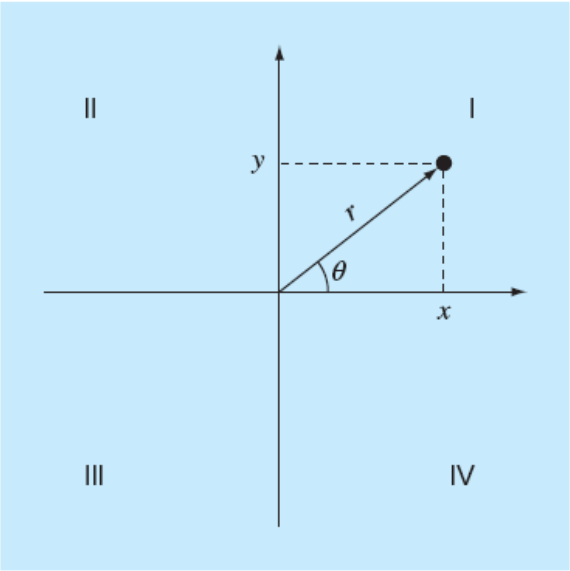


FIGURE P2.14

It is relatively straightforward to compute Cartesian coordinates (x, y) on the basis of polar coordinates (r, θ). The reverse process is not so simple. The radius can be computed by the following formula:

$$r = \sqrt{x^2 + y^2}$$

If the coordinates lie within the first and fourth coordinates (i.e., $x > 0$), then a simple formula can be used to compute θ

$$q = \tan^{-1}\left(\frac{y}{x}\right)$$

The difficulty arises for the other cases. The following table summarizes the possibilities:

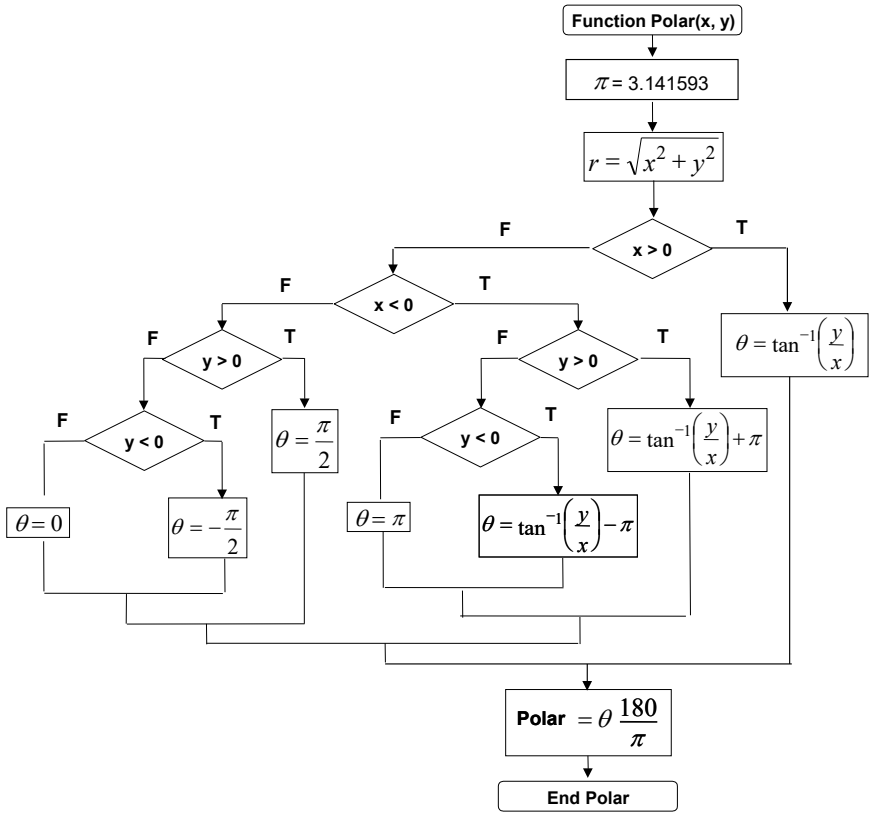
x	*y*	*θ*
<0	>0	$\tan^{-1}(y/x) + p$
<0	<0	$\tan^{-1}(y/x) - p$
<0	=0	p
=0	>0	$p/2$
=0	<0	$-p/2$
=0	=0	0

(a) Write a well-structured flowchart for a subroutine procedure to calculate r and θ as a function of x and y. Express the final results for θ in degrees.

(b) Write a well-structured function procedure based on your flowchart. Test your program by using it to fill out the following table:

x	*y*	*r*	*θ*
1	0		
1	1		
0	1		
−1	1		
−1	0		
−1	−1		
0	−1		
1	−1		
0	0		

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 as in the following script:

```
clc; format compact
x=[1 1 0 -1 -1 -1 0 1 0];
y=[0 1 1 1 0 -1 -1 -1 0];
polar(x,y)
```

When the script is run, the resulting output is

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

2.15 Develop a well-structured function procedure that is passed a numeric grade from 0 to 100 and returns a letter grade according to the following scheme:

Letter	Criteria
A	90 ≤ numeric grade ≤ 100
B	80 ≤ numeric grade < 90
C	70 ≤ numeric grade < 80
D	60 ≤ numeric grade < 70
F	numeric grade < 60

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

VBA Function Procedure

```
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
```

MATLAB M-File

```
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 Develop well-structured function procedures to determine **(a)** the factorial; **(b)** the minimum value in a vector; and **(c)** the average of the values in a vector.

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

(a) Factorial

VBA Function Procedure

```
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
```

MATLAB M-File

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

(b) Minimum

VBA Function Procedure

```
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
```

MATLAB M-File

```
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

VBA Function Procedure

```
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
```

MATLAB M-File

```
function xm = xmean(x)
n = length(x);
s = x(1);
for i = 2:n
  s = s + x(i);
end
xm = s / n;
```

2.17 Develop well-structured programs to **(a)** determine the square root of the sum of the squares of the elements of a two-dimensional array (i.e., a matrix) and **(b)** normalize a matrix by dividing each row by the maximum absolute value in the row so that the maximum element in each row is 1.

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

(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

```
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
```

2.18 *Piecewise functions* are sometimes useful when the relationship between a dependent and an independent variable cannot be adequately represented by a single equation. For example, the velocity of a rocket might be described by

$$v(t) = \begin{cases} 11t^2 - 5t & 0 \le t \le 10 \\ 1100 - 5t & 10 \le t \le 20 \\ 50t + 2(t-20)^2 & 20 \le t \le 30 \\ 1520e^{-0.2(t-30)} & t > 30 \\ 0 & \text{otherwise} \end{cases}$$

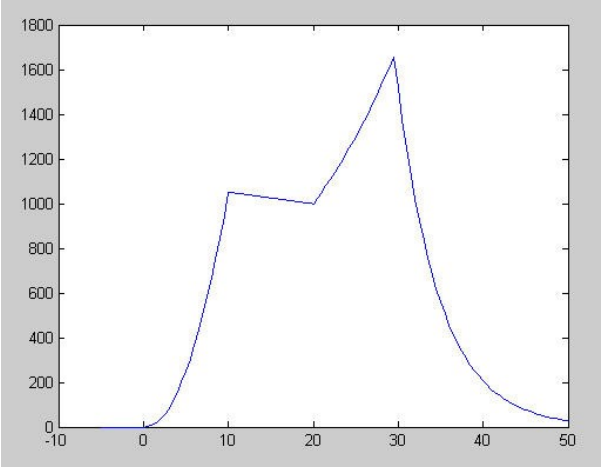
Develop a well-structured function to compute v as a function of t. Then use this function to generate a table of v versus t for $t = -5$ to 50 at increments of 0.5.

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 Develop a well-structured function to determine the elapsed days in a year. The function should be passed three values: `mo` = the month (1–12), `da` = the day (1–31), and `leap` = (0 for non–leap year and 1 for leap year). Test it for January 1, 1999; February 29, 2000; March 1, 2001; June 21, 2002; and December 31, 2004. *Hint:* A nice way to do this combines the `for` and the `switch` structures.

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 Develop a well-structured function to determine the elapsed days in a year. The first line of the function should be set up as

```
function nd = days(mo, da, year)
```

where `mo` = the month (1–12), `da` = the day (1–31), and `year` = the year. Test it for January 1, 1999; February 29, 2000; March 1, 2001; June 21, 2002; and December 31, 2004.

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 Manning's equation can be used to compute the velocity of water in a rectangular open channel,

$$U = \frac{\sqrt{S}}{n}\left(\frac{BH}{B + 2H}\right)^{2/3}$$

where U = velocity (m/s), S = channel slope, n = roughness coefficient, B = width (m), and H = depth (m). The following data are available for five channels:

n	S	B	H
0.036	0.0001	10	2
0.020	0.0002	8	1
0.014	0.0012	19	1.7
0.030	0.0007	24	3
0.021	0.0004	15	2.6

Write a well-structured program that computes the velocity for each of these channels. Have the program display the input data along with the computed velocity in tabular form where velocity is the fifth column. Include headings on the table to label the columns.

A MATLAB M-file can be written as

```
function Manning(A)
% computes the velocity of water in an open rectangular channel via
% Mannings equation for each row of matrix A
% Inputs:   Matrix A of dimension r x 4 with columns information
%           column 1 n roughness coefficient
%           column 2 S channel slope
%           column 3 B width in meters
%           column 4 H depth in meters
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 called from a script to create the table,

```
% Chapter2 problem 21 calling script
A = [.036 .0001 10 2
.020 .0002 8 1
.014 .0012 19 1.7
.03 .0007 24 3
.021 .0004 15 2.6];
Manning(A)
```

Running the script from the MATLAB command line:

```
>> ch2p21

    n         S          B          H          U
   0.036   0.0001      10.00       2.00     0.3523
   0.020   0.0002       8.00       1.00     0.6094
   0.014   0.0012      19.00       1.70     3.1581
   0.030   0.0007      24.00       3.00     1.5809
   0.021   0.0004      15.00       2.60     1.4767
```

2.22 A simply supported beam is loaded as shown in Fig. P2.22. Using singularity functions, the displacement along the beam can be expressed by the equation

$$u_y(x) = \frac{-5}{6}\left[\langle x-0\rangle^4 - \langle x-5\rangle^4\right] + \frac{15}{6}\langle x-8\rangle^3 + 75\langle x-7\rangle^2 + \frac{57}{6}x^3 - 238.25x$$

By definition, the singularity function can be expressed as follows:

$$\langle x-a\rangle^n = \begin{cases} (x-a)^n & \text{when } x > a \\ 0 & \text{when } x \le a \end{cases}$$

Develop a program that creates a plot of displacement versus distance along the beam x. Note that $x = 0$ at the left end of the beam.

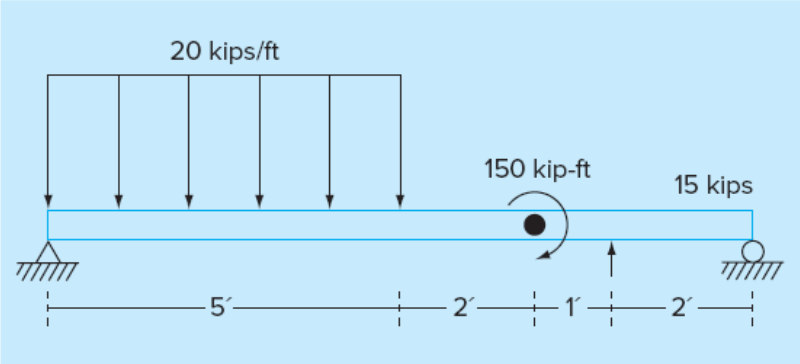


FIGURE P2.22

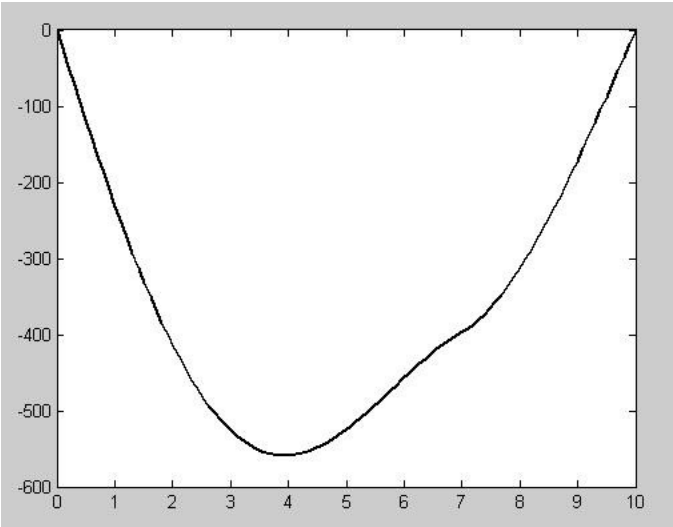
A MATLAB M-file can be written as

```
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 The volume V of liquid in a hollow horizontal cylinder of radius r and length L is related to the depth of the liquid h by

$$V = \left[r^2 \cos^{-1}\left(\frac{r-h}{r}\right) - (r-h)\sqrt{2rh-h^2}\right]L$$

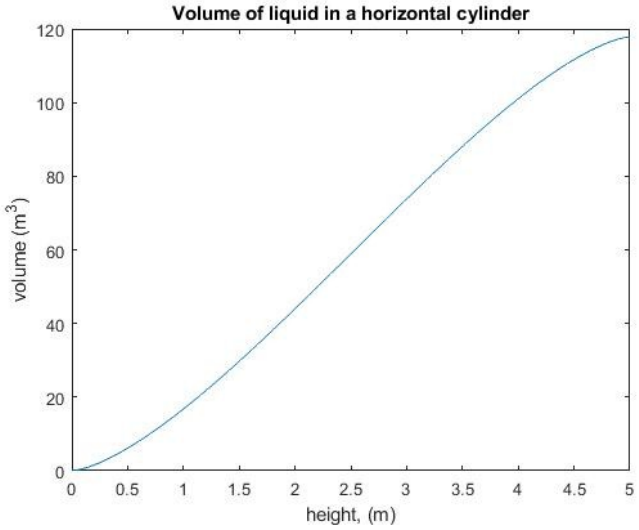
Develop a well-structured function to create a plot of volume versus depth. Test the program for $r = 2.5$ m and $L = 6$ m.

A MATLAB M-file can be written as

```
function cylinder(r, L)
% calculates the volume of water in horizontal cylinder of radius r and
% length L, in meters
% Ouptut: plot of Volume (m^3) versus height (m)
h = linspace(0,2*r);
V = (r^2*acos((r-h)./r)-(r-h).*sqrt(2*r*h-h.^2))*L;
plot(h, V)
title('Volume of liquid in a horizontal cylinder')
xlabel('height, (m)')
ylabel('volume (m^3)')
```

This function can be run to create the plot,

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



2.24 Develop a well-structured program to compute the velocity of a parachutist as a function of time using Euler's method. Test your program for the case where $m = 85$ kg and $c = 11$ kg/s. Perform the calculation from $t = 0$ to 20 s with a step size of 2 s. Use an initial condition that the parachutist has an upward velocity of 25 m/s at $t = 0$. At $t = 10$ s, assume that the parachute is instantaneously deployed so that the drag coefficient jumps to 55 kg/s.

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
% calculates the velocity of parachutist using Euler method and parameters
% from Chapra 8e Chapter 2, problem 24
g = 9.81;
m = 85; c = 11;
ti = 0; tf = 20; dt = 2;
vi = -25;
tc = 10; cc = 55;
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
figure('Position',[300 300 800 400])
plot(tout,vout)
title('Velocity of parachutist, chute deployed at t = 10 s')
xlabel('velocity (m/s)')
ylabel('time (s)')
saveas(gcf,'ch2p24.jpg')
z=[tout;vout];
fprintf('    t        v\n');
fprintf('%5d %10.3f\n',z);
```

```
>> parachute
    t        v
    0    -25.000
    2      1.091
    4     20.428
    6     34.761
    8     45.384
   10     53.258
   12      3.956
   14     18.456
   16     14.192
   18     15.446
   20     15.077
```

2.25 The pseudocode in Fig. P2.25 computes the factorial. Express this algorithm as a well-structured function in the language of your choice. Test it by computing 0! and 6!. In addition, test the error trap by trying to evaluate −2!.

```
FUNCTION fac(n)
IF n ≥ 0 THEN
  x = 1
  DOFOR i = 1, n
    x = x · i
  END DO
  fac = x
ELSE
  display error message
  terminate
ENDIF
END fac
```

FIGURE P2.25

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

MATLAB example script to generate test fac.m

```
% Chapra 8e, Chapter 2, problem 25
format compact
fac(0)
fac(6)
fac(-2)

>> ch2p25
ans =
     1
ans =
   720
```

```
Error using fac (line 10)
value must be positive
Error in ch2p25 (line 5)
fac(-2)
```

The first two lines are the error generated from the function file, and the second two lines are from the script file.

2.26 The height of a small rocket y can be calculated as a function of time after blastoff with the following piecewise function:

$$y = 38.1454t + 0.13743t^3 \qquad 0 \le t < 15$$

$$y = 1036 + 130.909(t - 15) + 6.18425(t - 15)^2 - 0.428(t - 15)^3 \qquad 15 \le t < 33$$

$$y = 2900 - 62.468(t - 33) - 16.9274(t - 33)^2 + 0.41796(t - 33)^3 \qquad t \ge 33$$

Develop a well-structured pseudocode function to compute y as a function of t. Note that if the user enters a negative value of t or if the rocket has hit the ground ($y \le 0$), then a value of zero is returned for y. Also, the function should be invoked in the calling program as `height(t)`. Write the algorithm **(a)** as pseudocode or **(b)** in a high-level language of your choice.

(a) Pseudocode:

```
FUNCTION height(t)
IF t < 0 THEN
  y = 0
ELSE IF t < 15 THEN
  y = 38.1454t + 0.13743t^3
ELSE IF t < 33 THEN
  y = 1036   + 130.909(t - 15) + 6.18425(t - 15)^2 - 0.428 (t - 15)^3
ELSE
  y = 2900   - 62.468(t - 33) - 16.9274(t - 33)^2 + 0.41796 (t - 33)^3
END IF
IF y < 0 THEN y = 0
height = y

END
```

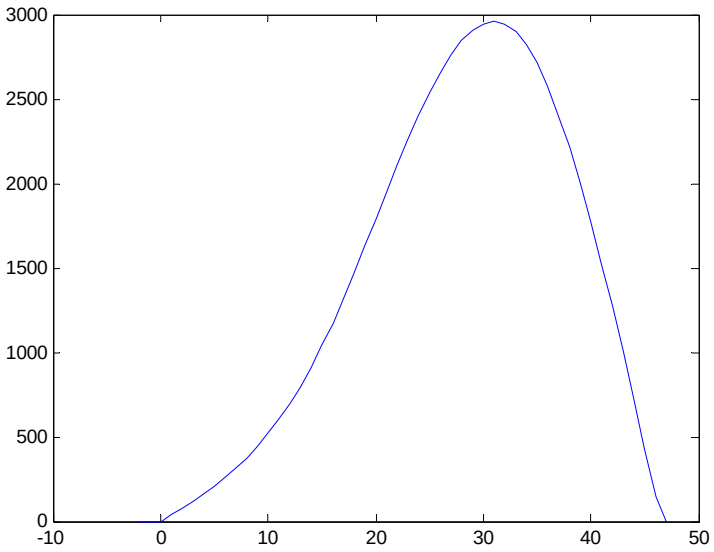
(b) MATLAB:

```
function y = height(t)
%Function to compute height of rocket from piecewise function
%   y = height(t)
% input:
%   t = time
% output:
%   y = height

if  t < 0
  y = 0;
elseif  t < 15
  y = 38.14544*t + 0.137428*t^3;
elseif  t < 33
  y = 1036 + 130.909*(t - 15) + 6.18425*(t - 15)^2 - 0.428*(t - 15)^3;
else
  y = 2900 - 62.468*(t - 33) - 16.9274*(t - 33)^2 + 0.41796*(t - 33)^3;
end
if  y < 0, y = 0; end
end
```

Here is a script that uses the function to generate a plot:

```
clc,clf
t=[-2:47];
for i=1:length(t)
  y(i)=height(t(i));
end
plot(t,y)
```



VBA:

```
Option Explicit

Function height(t)
If t < 0 Then
  y = 0
ElseIf t < 15 Then
  y = 38.14544 * t + 0.137428 * t ^ 3
ElseIf t < 33 Then
  y = 1036 + 130.909 * (t - 15) + 6.18425 * (t - 15) ^ 2 _
      - 0.428 * (t - 15) ^ 3
Else
  y = 2900 - 62.468 * (t - 33) - 16.9274 * (t - 33) ^ 2 _
       + 0.41796 * (t - 33) ^ 3
End If
If y < 0 Then y = 0
height = y
End Function
```

2.27 As depicted in Fig. P2.27, a water tank consists of a cylinder topped by the frustum of a cone. Develop a well-structured function in a high-level language or macro language of your choice to compute the volume given the water level h (m) above the tank's bottom. Design the function so that it returns a value of zero for negative h's and the value of the maximum filled volume for h's greater than the tank's maximum depth. Given the following parameters, $H_1 = 11$ m, $r_1 = 3.5$ m, $H_2 = 5$ m, and $r_2 = 6$ m, test your function by using it to compute the volumes and generate a graph of the volume as a function of level from $h = -1$ to 17 m.

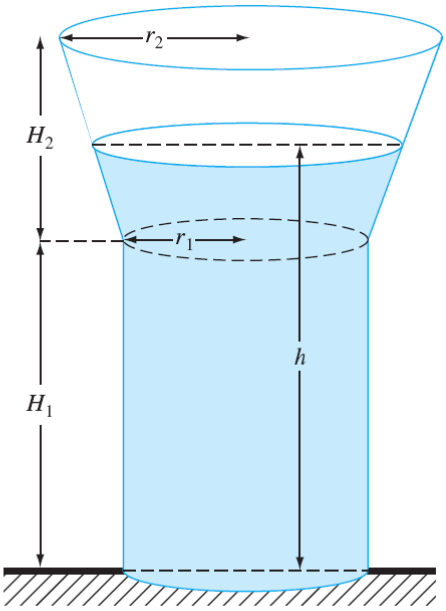


FIGURE P2.27

We must first identify the general formulas for the volumes. For example, for the full cylinder

$$V = \pi r_1^2 H_1 \tag{1}$$

and for the volume of the full circular cone frustum

$$V = \frac{\pi H_2}{3}\left(r_1^2 + r_2^2 + r_1 r_2\right) \tag{2}$$

With this knowledge we can come up with the other cases that can occur:

Case 1: Full tank or overflowing tank.

$$V = \pi r_1^2 H_1 + \frac{\pi H_2}{3}\left(r_1^2 + r_2^2 + r_1 r_2\right)$$

Case 2: The depth, $h \le 0$. $V = 0$

Case 3: Partially-full cylinder ($0 < h < H_1$)

$$V = \pi r_1^2 h$$

Case 4: Full cylinder with partially-full frustum ($H_1 \le h < H_1 + H_2$)

$$V = \pi r_1^2 H_1 + \frac{\pi (h - H_1)}{3}\left(r_1^2 + r_2(h)^2 + r_1 r_2(h)\right)$$

where $r_2(h)$ = the radius of the top of the partially-filled frustum. This quantity can be computed using the problem parameters via linear interpolation as

$$r_2(h) = r_1 + \frac{r_2 - r_1}{H_2}\left(h - H_1\right)$$

We can then use an if/then/elseif control structure to logically combine these cases as in

$$V = \pi r_1^2 H_1 + \frac{\pi H_2}{3}\left(r_1^2 + r_2^2 + r_1 r_2\right)$$

IF $h \le 0$ *THEN*

$V = 0$

ELSEIF $h < H_1$ *THEN*

$V = \pi r_1^2 h$

ELSEIF $h < H_1 + H_2$ *THEN*

$$r_2(h) = r_1 + \frac{r_2 - r_1}{H_2}\left(h - H_1\right)$$

$$V = \pi r_1^2 H_1 + \frac{\pi(h - H_1)}{3}\left(r_1^2 + r_2(h)^2 + r_1 r_2(h)\right)$$

ENDIF

Notice how Eqs. (1) and (2) are used several times, but with different arguments. This suggests that we should represent them as independent functions that would be called by the main function. We do this in the following code.

VBA/Excel.

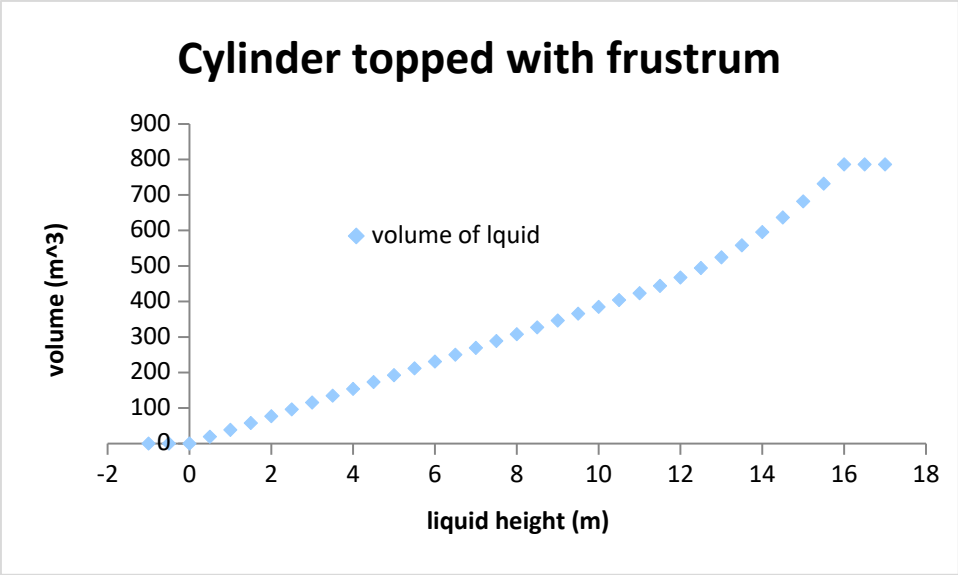
```
Option Explicit

Const pi As Double = 3.14159265358979

Function Vol(h, r1, h1, r2, h2)
Dim r2h As Double
Vol = VCyl(r1, h1) + VFus(r1, r2, h2)
If h <= 0 Then
  Vol = 0
ElseIf h < h1 Then
  Vol = VCyl(r1, h)
ElseIf h < h1 + h2 Then
  r2h = r1 + (r2 - r1) / h2 * (h - h1)
  Vol = VCyl(r1, h1) + VFus(r1, r2h, h - h1)
End If
End Function

Function VCyl(r, y)
VCyl = pi * r ^ 2 * y
End Function

Function VFus(r1, r2, h2)
VFus = pi * h2 / 3 * (r1 ^ 2 + r2 ^ 2 + r1 * r2)
End Function
```



MATLAB. Written as a function with subfunctions:

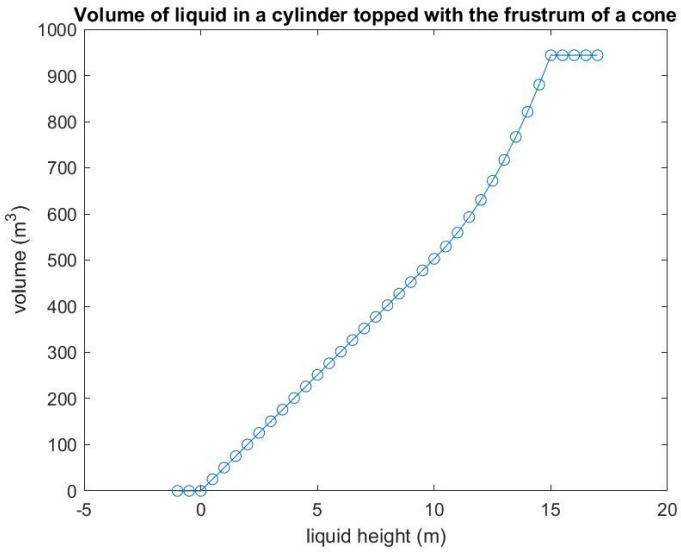
```
function V=Vol(h, r1, h1, r2, h2)
V = VCyl(r1, h1) + VFus(r1, r2, h2);
if h <= 0
  V = 0;
elseif h < h1
  V = VCyl(r1, h);
elseif h < h1 + h2
  r2h = r1 + (r2 - r1) / h2 * (h - h1);
  V = VCyl(r1, h1) + VFus(r1, r2h, h - h1);
end
end

function V=VCyl(r, y)
V = pi * r ^ 2 * y;
end

function V=VFus(r1, r2, h2)
V = pi * h2 / 3 * (r1 ^ 2 + r2 ^ 2 + r1 * r2);
end
```

Here is a script that uses the functions to develop a plot of volume versus height:

```
clc,clf, clear
h=[-1:0.5:17];
r1=4; H1=10; r2=6.5; H2=5;
n=length(h);
lvol=zeros(1,n);
for i=1:n
  lvol(i)=Vol(h(i),r1, H1, r2, H2);
end
plot(h,lvol, 'o-')
title('Volume of liquid in a cylinder topped with the frustrum of a cone')
xlabel('liquid height (m)')
ylabel('volume (m^3)')
saveas(gcf,'ch2p27.jpg')
```



2.28 Write a well-structured function procedure named `Fnorm` to calculate the Frobenius norm of an $m \times n$ matrix using nested count-controlled (for) loops:

$$\|A\|_f = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}^2}$$

Test your function with

$$A = \begin{bmatrix} 5 & 7 & 9 \\ 1 & 8 & 4 \\ 7 & 6 & 2 \end{bmatrix}$$

MATLAB function code

```
function out = Fnorm(A)
[rmax,cmax] = size(A);
sum2 = 0;
for idx = 1:rmax
    for jdx = 1:cmax
        sum2 = sum2 + A(idx,jdx)^2;
    end
end
out = sqrt(sum2);
```

MATLAB calling script

```
% Chapra 8e, Chapter 2, problem 28
A = [5 7 9
1 8 4
7 6 2];
fprintf('Frobineous norm of A is: %7.5f\n',Fnorm(A))
% cross-check with built-in function
norm(A,'fro')
```

```
>> ch2p28
Frobineous norm of A is: 18.02776
ans =
   18.0278
>>
```

2.29 The pressure and temperature of the atmosphere are constantly changing depending on a number of factors including altitude, latitude/longitude, time of day, and season. To take all these variations into account when considering the design and performance of flight vehicles is impractical. Therefore, a *standard atmosphere* is frequently used to provide engineers and scientists with a common reference for their research and development. The *International Standard Atmosphere* is one such model of how conditions of the earth's atmosphere change over a wide range of altitudes, or elevations. The following table shows values of temperature and pressure at selected altitudes.

Layer index i	Layer name	Base geopotential altitude above MSL h (km)	Lapse rate (°C/km)	Base temperature T (°C)	Base pressure p (Pa)
1	Troposphere	0	−6.5	15	101325
2	Tropopause	11	0	−56.5	22632
3	Stratosphere	20	1	−56.5	5474.9
4	Stratosphere	32	2.8	−44.5	868.02
5	Stratopause	47	0	−2.5	110.91
6	Mesosphere	51	−2.8	−2.5	66.939
7	Mesosphere	71	−2.0	−58.5	3.9564
8	Mesopause	84.852	—	−86.28	0.3734

The temperature at each altitude can then be computed as

$$T(h) = T_i + \gamma_i (h - h_i) \qquad h_i < h \le h_{i+1}$$

where $T(h)$ = temperature at altitude h (°C), T_i = the base temperature for layer i (°C), γ_i = lapse rate, or the rate at which atmospheric temperature decreases linearly with increase in altitude for layer i (°C/km), and h_i = base geopotential altitude above mean sea level (MSL) for layer i. The pressure at each altitude can then be computed as

$$p(h) = pi + \frac{p_{i+1} - p_i}{h_{i+1} - h_i}(h - h_i)$$

where $p(h)$ = pressure at altitude h (Pa ≡ N/m^2), p_i = the base pressure for layer i (Pa). The density, ρ (kg/m^3), can then be calculated according to a molar form of the *ideal gas law:*

$$r = \frac{pM}{RT_a}$$

where M = molar mass (≅ 0.0289644 kg/mol), R = the universal gas constant (8.3144621 J/(mol K), and T_a = absolute temperature (K) = T + 273.15.

Develop a function, `StdAtmDens`, to determine values of the density for a given altitude. If the user requests a value outside the range of altitudes, have the function display an error message and terminate the application. Test your function for altitudes of −200, 0, 11, 40, 84.852, and 100 km.

Note there is no test data provided in the problem.
The solution presented here affords an opportunity to introduce students to structures. The overhead in setting up the structure array results in a very simple calculation section once the correct layer is identified. This complexity is not required to solve the problem as presented in the text. The code here would need to be modified to work with vector input; however, this would also be true for most solution approaches.

MATLAB function code

```
function [rho p T LName] = StdAtmDens(h)
% returns the denisty of air in the standard atmosphere given a height
% above sea level in km
% optionally also returns pressure and temperature at that height
%{
Chapra 8e Problem 2.29
RJ Genik II, 8 December 2019
The solution developed here provides an example of a structured array. This
would usually be filled from a text file, but here we use input arrays
because the data table is small.
Revised 12 December 2019 - corrected opensedclo interval checks as
h values allowed h > hmin, h <= hmax
%}
R = 8.3144621; % universal gas constant in J/(mol K)
M = 0.0289644; % molar mass in kg/mol
layernum = [1:8];
layername = {'Troposphere' 'Tropopause' 'Stratosphere' 'Stratosphere'...
    'Stratopause' 'Mesosphere' 'Mesosphere' 'Mesopause'};
htable = [0 11 20 32 47 51 71 84.852];% km
lpsrate = [-6.5 0 1 2.8 0 -2.8 -2.0 NaN];% deg C / km
baseT = [15 -56.5 -56.5 -44.5 -2.5 -2.5 -58.5 -86.28]; % deg C
baseP = [101325 22632 5474.9 868.02 110.91 66.939 3.9564 0.3734];% Pa
% build data structure to hold tabular data - could be read from file
% instead, as mentioned above
for idx = 1:length(layernum)
    atmos(idx).LayerIndex = layernum(idx);
    atmos(idx).LayerName = layername(idx);
    atmos(idx).hbase = htable(idx);
    atmos(idx).gamma = lpsrate(idx);
    atmos(idx).Tbase = baseT(idx);
    atmos(idx).pbase = baseP(idx);
end
hmax = atmos(end).hbase;
hmin = atmos(1).hbase;
if (h<=hmin) || (h>hmax)
    error('input height %g outside of model range %g < h <= %g (km)\n', ...
        h, hmin, hmax)
else
    % find the appropriate layer
    for jdx = 1:length(atmos)
        if h>atmos(jdx).hbase
            ilayer =  jdx;
        end
    end
end
LName = atmos(ilayer).LayerName;
% Calculate temperature, pressure, and density
hrel = h - atmos(ilayer).hbase;
T = atmos(ilayer).Tbase + atmos(ilayer).gamma*hrel;
p = atmos(ilayer).pbase + (atmos(ilayer+1).pbase - atmos(ilayer).pbase)...
    / (atmos(ilayer+1).hbase - atmos(ilayer).hbase) * hrel;
rho = p*M/(R*(T + 273.15));
```

MATLAB error test from command line

```
>> StdAtmDens(-200)
Error using StdAtmDens (line 36)
input height -200 outside of model range 0 < h <= 84.852 (km)
>> StdAtmDens(0)
Error using StdAtmDens (line 36)
input height 0 outside of model range 0 < h <= 84.852 (km)
>> StdAtmDens(100)
```

```
Error using StdAtmDens (line 36)
input height 100 outside of model range 0 < h <= 84.852 (km)
>>
```

Similarly, the other input data that is not supposed to generate an error can be run to yield the results:

MATLAB test data

height (km)	r (kg/m^3)	T(C)	P (Pa)
11	0.36391	-56.5	22632
40	0.0064417	-22.1	464.228
84.9	6.96E-06	-86.2	0.3734

Finally, one can write a test script that will catch errors in a more controlled manner, and output results for all of the requested inputs. For neatness, the error generating values have been moved to the end of the input vector, but this is not required. This example afford the opportunity to practice MATLAB's try/catch structure for error handling, and impart to students that better error messages inform the user why calculations failed.

MATLAB test script:

```
% Chapra 8e Chapter 2 problem 29 test data
clear; clc
htest = [11, 40, 84.852, 100, -200, 0];

ltest = length(htest);

  fprintf('height(km)    Layer Name   rho(kg/m^3) T(deg C)   P(Pa)\n')
for idx =  1:ltest
  try
    [dens pres temp LName] = StdAtmDens(htest(idx));
    fprintf('   %5.1f  %15s %12.5g    %5.1f   %g\n',htest(idx), ...
        LName{1},dens,temp, pres)
  catch ME
      fprintf(2, '** %5.1f     *** function returned error ***\n', htest(idx))
      fprintf(2,'** %60s\n',ME.message)
  end
end
>> ch2p29
height(km)    Layer Name   rho(kg/m^3) T(deg C)   P(Pa)
    11.0      Troposphere      0.36391    -56.5   22632
    40.0     Stratosphere    0.0064417    -22.1   464.228
    84.9       Mesosphere   6.9581e-06    -86.2   0.3734
** 100.0     *** function returned error ***
** input height 100 outside of model range 0 < h <= 84.852 (km)

** -200.0     *** function returned error ***
** input height -200 outside of model range 0 < h <= 84.852 (km)

**   0.0     *** function returned error ***
**  input height 0 outside of model range 0 < h <= 84.852 (km)
```

2.30 Develop a function to convert a vector of temperatures from Celsius to Fahrenheit and vice versa. Test it with the following data for the average monthly temperatures at Death Valley, CA, and at the South Pole.

Day		15	45	75	105	135	165	195	225	255	285	315	345
Death Valley	°F	54	60	69	77	87	96	102	101	92	78	63	52
South Pole	°C	−27	−40	−53	−56	−57	−57	−59	−59	−59	−50	−38	−27

Using a single function for both conversions requires a control mechanism. One method is to accept text input that contains a C or F and key the calculation on finding the input unit. Another method is to accept two inputs, one the temperature and the second a text code; in this method, an error is thrown if the code is not understood. A final method is to default to one calculation and accept an optional second argument to reverse the calculation.

The second method mentioned above is shown here as a VBA function. This VBA function does not directly generate vector output from vector input and requires calling from a range of spreadsheet cells.

```
Option Explicit

Function TempConv(Tin, cCode)

If UCase(cCode) = "C2F" Then
    TempConv = 9 / 5 * Tin + 32
ElseIf UCase(cCode) = "F2C" Then
    TempConv = (Tin - 32) * 5 / 9
Else
    MsgBox ("invalid control code, F2C or C2F only")
    TempConv = "Err"
End If

End Function
```

Similar results can be obtained with a MATLAB function using a flag value to reverse the calculation. A character indicating units returned is used to allow users to verify the calculation proceeded as intended.

```
function [Tout, Tunits] = FtoC(Tin, rFlag)
if rFlag~=0
    Tout = 9/5*Tin + 32;
    Tunits = 'F';
else
    Tout = (Tin - 32)*5/9;
    Tunits = 'C';
end
```

Called from a script with the vector input, tabular results are obtained:

```
% Chapra 8e Chapter 2, problem 30
Day = [15 45 75 105 135 165 195 225 255 285 315 345];
DeathF = [54 60 69 77 87 96 102 101 92 78 63 52];
SouthC = [-27 -40 -53 -56 -57 -57 -59 -59 -59 -50 -38 -27];
DeathC = FtoC(DeathF,0);
SouthF = FtoC(SouthC, 1);
fprintf('Day             ')
fprintf('%5.0f', Day); fprintf('\n')
fprintf('Death Valley °F ')
fprintf('%5.0f', DeathF); fprintf('\n')
fprintf('Death Valley °C ')
fprintf('%5.0f', DeathC); fprintf('\n')
```

```
fprintf('South Pole °C   ')
fprintf('%5.0f', SouthC); fprintf('\n')
fprintf('South Pole °F   ')
fprintf('%5.0f', SouthF); fprintf('\n')
```

Results agree from both procedures, rounded to nearest degree (Note: results are rounded to fit on the page neatly in the table below – the results agree between the two methods to full machine precision):

Day		15	45	75	105	135	165	195	225	255	285	315	345
Death Valley	°F	54	60	69	77	87	96	102	101	92	78	63	52
	°C	12	16	21	25	31	36	39	38	33	26	17	11
South Pole	°C	-27	-40	-53	-56	-57	-57	-59	-59	-59	-50	-38	-27
	°F	-17	-40	-63	-69	-71	-71	-74	-74	-74	-58	-36	-17

2.31 As depicted in Fig. P2.31, the downward deflection, y (m), of a cantilever beam with a uniform load, w (kg/m), can be computed as

$$y = \frac{w}{24EI}(x^4 - 4Lx^3 + 6L^2x^2)$$

where x = distance (m), E = the modulus of elasticity = 2×10^{11} Pa, I = moment of inertia = 3.25×10^{-4} m^4, w = 10,000 N/m, and L = length = 4 m. This equation can be differentiated to yield the slope of the downward deflection as a function of x,

$$\frac{dy}{dx} = \frac{2}{24EI}(4x^3 - 12Lx^2 + 12L^2x)$$

If $y = 0$ at $x = 0$, use this equation with Euler's method ($\Delta x = 0.125$ m) to compute the deflection from $x = 0$ to L. Develop a plot of your results along with the analytical solution computed with the first equation.

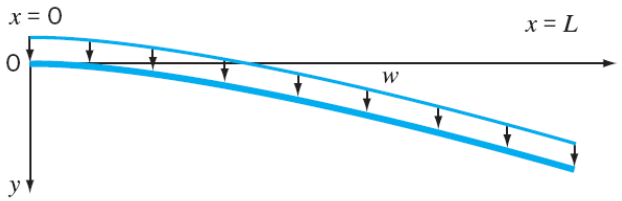


FIGURE P2.31

The problem definition does not include a specific value for the uniform load, but the shape of the curve is the same for every uniform load; therefore, we can plot the horizontal deflection per unit load as a function of distance along the cantilever.

MATLAB Script

```
% Chapra 8e, Chapter 2, problem 31
% RJ Genik II, 8 - 12 December 2019

clear; clf
I = 3.25E-4;% m^4
E = 2E+11; % Pa
L=4;% m
w= 10000;% N/m
y0 = 0;
```

```
ytrue = @(x) w/(24*E*I)*(x.^4 - 4*L*x.^3 + 6*L^2*x.^2) ; %kg^-1
dydxw = @(x) w/(24*E*I)*(4*x^3 - 12*L*x^2 + 12*L^2*x) ; %kg^-1/m

dx= 0.125; % m
x = 0:dx:L;
ywtrue = ytrue(x);
yw = zeros(1,length(x));
yw(1) = y0;
for idx = 1:length(x)-1
    yw(idx+1) = yw(idx) + dydxw(x(idx))*dx;
end
plot (x,ywtrue,'-b')
set(gca,'YDir','reverse','YLim',[-5E-4 5E-3])
hold on
plot (x,yw,'+g')
legend('Analytical Deflection', 'Euler Deflection')
xlabel('distance along cantilever (m)')
ylabel('horizontal deflection (m)')
title('Cantilever beam under uniform load 10^4 N/m')
saveas(gcf,'ch2p31.jpg')
```