

```

1: *****
2: /exercise4/factorial.f90
3: /exercise4/factorial.f90
4:
5: PROGRAM factorial
6: !*** calculate the first factorials of the numbers 1 -> 8
7:
8: IMPLICIT NONE
9:
10: INTEGER :: a, b=1
11:
12: DO a = 1, 8
13:   b = a*b
14:   PRINT*, 'factorial ', a, ' = ', b
15: END DO
16:
17: END PROGRAM factorial
18:
19: *****
20:
21: *****
22: /exercise5/nested_loop.f90
23: /exercise5/nested_loop.f90
24:
25: PROGRAM nested_loop
26: !*** Program to demonstrate nested loops
27: !**
28: !**
29:
30: IMPLICIT NONE
31:
32: INTEGER :: loop1, loop2, ans=0
33:
34: DO loop1=10,19
35:   DO loop2=21,30
36:     ans=ans+loop1*loop2
37:   END DO
38: END DO
39:
40: PRINT*, "Answer is", ans
41:
42: !** Ans = 36975
43:
44: END PROGRAM nested_loop
45:
46: *****
47:
48: *****
49: /exercise2/quadratic_real.f90
50: /exercise2/quadratic_real.f90
51: *****
52: *****
53: !**
54: !** PROGRAM: Quad
55: !**
56: !** PURPOSE: Examine a quadratic equation
57: !**
58: !**
59:
60: PROGRAM quad
61:
62: !**
63: !** Program to investigate a quadratic equation (y=a*x*x+b*x+c)
64: !**
65:
66: IMPLICIT NONE !** Force explicit declaration of all variables
67:

```

```

68: !** Declare required variables
69: LOGICAL :: check
70: REAL :: a=-3,b=6,c=1 !** Define the quadratic a*x*x+b*x+c
71: REAL :: ans1, ans2 !** Declare two variables to hold the two roots
72: REAL :: xturn !** Declare variable to hold the x value of turning point
73: REAL :: yturn !** Declare variable to hold the y value of turning point
74: REAL :: discrim !** Variable to validate quadratic
75: REAL :: realpart !** Hold real part of complex number
76: REAL :: imagpart !** Hold imaginary part of complex number
77:
78:
79: PRINT*, "Program to find the roots of a quadratic equation"
80: PRINT*, "a=", a, " b=", b, " c=", c
81:
82: check= .NOT. a==0 !** Set check to .TRUE. if a valid quadratic
83:
84: !**
85: !** Section to calculate root(s) of the quadratic
86: !**
87:
88: IF (check) THEN
89:
90:   discrim=b**2-4*a*c !** Calculate the discriminant of the quadratic.
91:
92:   IF (discrim > 0) THEN !** If real solutions exist then enter construct
93:
94:     PRINT*, "There are a two unique roots"
95:     ans1=(-b+sqrt(discrim))/(2*a) !** Calculate the first root
96:     ans2=(-b-sqrt(discrim))/(2*a) !** Calculate the second root
97:
98:     PRINT*, "Root One =", ans1 !** Output the first root to the screen
99:     PRINT*, "Root Two =", ans2 !** Output the second root to the screen
100:
101:   ELSEIF (discrim == 0) THEN
102:     PRINT*, "There is a single repeated root"
103:     ans1=-b/(2*a) ; ans2=ans1
104:     PRINT*, "Root =", ans1
105:   ELSE
106:     PRINT*, "No Real roots to this quadratic"
107:     PRINT*, "Complex roots are"
108:     realpart=-b/(2*a)
109:     imagpart=SQRT(-discrim)/(2*a)
110:     PRINT*, "Root One =", realpart, "+", imagpart, "i"
111:     PRINT*, "Root Two =", realpart, "-", imagpart, "i"
112:   ENDIF
113: ELSE
114:   PRINT*, "This is not a valid quadratic"
115: ENDIF
116:
117: !**
118: !** Section to calculate the quadratic's turning point and it's nature
119: !**
120:
121: ! BUG IN CODE HERE -- BELOW WILL NOT WORK FOR COMPLEX ROOTS
122:
123: IF (check) THEN
124:   !** Calculate the quadratics turning point
125:   xturn=-b/(2*a)
126:   PRINT*, "xturn2=", (ans2+ans1)/2
127:   yturn=-b*b/(4*a)+c
128:   PRINT*, "Turning Point = (x,y) = (" , xturn, ",", yturn, ") "
129:
130: !** Calculate Max or Min point of
131: IF (a .LT. 0) THEN
132:   PRINT*, "turning point is a maximum"
133: ELSE
134:   PRINT*, "turning point is a minimum"

```

```

135:   ENDIF
136:   ENDIF
137:
138: END PROGRAM quad
139:
140: *****
141: *****
142:
143: *****
144: *****
145: ./exercise3/power3_loop.f90
146:
147: PROGRAM power3_loop
148: !** calculate the 3rd power of a for 1 <= a <= 8
149:
150: IMPLICIT NONE
151:
152: INTEGER :: a, b
153:
154: DO a = 1, 8
155:   b = a**a
156:   PRINT*,a,"to the power of three = ",b
157: END DO
158:
159: END PROGRAM power3_loop
160: *****
161: *****
162:
163: *****
164: *****
165: ./exercise6/array_example.f90
166:
167: PROGRAM array_example
168: !** Program to declare a 4 by 4 matrix and then
169: !** initialise such that each element is the sum of its row
170: !** index and twice its column index
171:
172: IMPLICIT NONE
173:
174: REAL, DIMENSION(4,4) :: array !** Create 2d array to represent a matrix
175: INTEGER :: i,j !** Define two loop variables
176:
177: DO i=1,4 !** Loop over rows
178:   Do j=1,4 !** Loop over columns
179:     array(i,j)=i+2*j !** Set matrix value
180:   END DO
181:   PRINT*, "ROW ",i,"=" ,array(i,:) !** Print each row to screen
182: END DO
183:
184: END PROGRAM array_example
185:
186: *****
187:
188: *****
189: *****
190: ./exercise1/logical_test2.f90
191:
192: PROGRAM logical_test2
193:
194: IMPLICIT NONE
195:
196: REAL :: height=1.85, weight=95.0
197: INTEGER :: age=55
198: LOGICAL :: drinker=.TRUE., employed=.TRUE., smoker=.TRUE.
199: LOGICAL :: test1,test2,test3
200:
201: If (age>50) Print*, "Over fifty"

```

```

202: test1 = employed .AND. (age<45)
203: test2 = smoker .AND. drinker .AND. (height<2.00)
204: test3 = (.NOT. smoker .OR. (age<=55) .AND. (weight>50)) .AND. (height<=1.84)
205:
206:
207: PRINT*,test1,test2,test3
208:
209: END PROGRAM logical_test2
210:
211: *****
212: *****
213:
214: *****
215: *****
216: ./exercise1/logical_test.f90
217:
218: PROGRAM logical_test
219:
220: IMPLICIT NONE
221:
222: LOGICAL :: tst1=.TRUE., tst2=.TRUE., tst3=.FALSE.
223: LOGICAL :: ans1,ans2,ans3
224:
225: ans1=tst1 .OR. tst2 .AND. tst3
226: ans2=(tst1 .OR. tst2) .AND. tst3
227: ans3=tst1 .OR. (tst2 .AND. tst3)
228:
229:
230: PRINT*, ans1,ans2,ans3
231:
232: END PROGRAM logical_test
233:
234: *****
235: *****
236:

```