Practice Problems

1. Write a Fortran program which finds the maximum value of n for which the value of $n!$ can be represented using the "standard" 32-bit integer numbers (it can be assumed that the largest 32-bit integer number is 2E9).

2. Write a Fortran function *HeapSort(A,n)* which returns ordered integer array $A[1:n]$ using the heap sort. Also, write a Fortran program which reads the data, invokes *HeapSort* and prints the results.

3. Write a Fortran function *Balance(A,n)* which returns an integer array $A[1:n]$ ordered in such a way, that the binary tree created from it is almost balanced (i.e., the heights of the left and right subtrees at each node of the tree can differ by not more than 1).

4. Modify the Gaussian elimination program (*LinearEqs2*) to be used for block–diagonal matrices (i.e., for the case when the solution is obtained for consecutive blocks, one after another).

5. Modify the solution of tridiagonal systems (*LinearEqs4*) to solve banded systems of linear equations composed of 5 diagonals.

6. A perfect number is a positive integer which is equal to the sum of its proper divisors, i.e., the sum of all divisors excluding the number itself (but including 1). The first perfect number is $6 = 3 + 2 + 1$; 28 is another perfect number. Write a Fortran logical function *Perfect(n)* which returns TRUE if n is a perfect number and FALSE otherwise.

   Hint: The function can generate the prime numbers not greater than $\sqrt{n}$, and use these numbers to check if n is perfect.

7. Write a Fortran logical function *MagicSquare(A,n)* which returns TRUE if the square matrix $A[1:n, 1:n]$ is "magic", i.e., the sums of all rows, all colums and the main diagonals are equal; otherwise FALSE is returned. Also, write a Fortran program which reads the data, invokes the function *MagicSquare* and prints the result.

8. Write a Fortran character function *FindDigit(s,n)* which returns the most frequent digit in the string s of length n. All characters which are not digits are ignored. If several digits occur in s the same number of times, the "last" one (in the sense of ordering) is returned. If s does not contain any digit, the question mark "?" should be returned.

   Hint: An integer array of 10 counters can be used to find the number of occurrences of each digit, and then to find and return the most frequent digit.

9. Write a Fortran string function *MyFormat (x,n)* which returns the "best" format for printing the real value x using n characters. For example, if $n = 10$ and $1.0 \le x < 10.0$, the returned string should be `(1X,F10.6)` (this string can be assigned to a character variable and then used in a PRINT/WRITE statement), for $10.0 \le x < 100.0$, the returned string should be `(1X,F10.5)`, and so on - implement not more than 5 different format cases. Make some reasonable assumptions about the details of generated formats (for example, do not output more than 7 significant digits even if n allows it). Write a Fortran program to test the *MyFormat* function, i.e., a program which reads the values of x and n, uses *Myformat* to generate the format string and then outputs x using this generated format to see how it works.

10. Encode the information is a text file by cyclically shifting all letters by k positions, so, for $k = 3$, 'a' becomes 'd', 'b' becomes 'e' and 'z' becomes 'c'. Write the result of encoding to another file.

    Note: Observe that another encoding by $26 - k$ positions converts the encoded text back to the original one.

11. Encode the information is a text file using a function $f : A \to A$ where A is the alphabet. The function f can be defined as a 26-character string with consecutive characters corresponding to consecutive letters. Check that no character is used more than once in this string (i.e., that the functions is 1:1). Write the result of encoding to another file and also find the reverse mapping which converts the encoded text back to the original one.

12. Define a derived type *Figure* to store information about three types of geometric figures: circle, square, and equilateral triangle (each figure requires only one numerical parameter). Write a subroutine *Order(A,n)* which orders the figures stored in array $A[1 : n]$ in the increasing order of the figure's area. Also write a Fortran program which reads the data, invokes the function *Order* and prints the results.

13. Write a Fortran function *Closure(A,n)* which returns the transitive closure of a graph represented by an adjacecy matrix $A[1 : n, 1 : n]$. Transitive closure of a graph is a square integer matrix $n \times n$ in which each entry $[i, j]$ is the length of the shortest path from "i" to "j" if such a path exists or 0 otherwise. Also write a Fortran program to read the data, invoke *Closure* and print the results.

14. Write a Fortran program which finds the number of arcs associated with each node of an undirected graph stored in a file as a list of arcs.

    Hint: Use an integer array for counting the arcs associated with each node.

15. Write a Fortran program which reads a description of a finite automaton (as a sequence of transitions, i.e., arcs labeled by input symbols), and checks if the automaton is minimal.

16. Write a Fortran program which reads a description of a finite automaton (as a sequence of transitions, i.e., arcs labeled by input symbols), checks if the automaton is deterministic, and performs the conversion if the original automaton is nondeterministic.

17. Modify the backtracking program (*FindingPath*) which finds a path between a pair of nodes in a directed graph to include the length of the path.

18. Modify the backtracking program (*FindingPath*) which finds a path between a pair of nodes in a directed graph to first find the length of the shortest path between these nodes and then to find the path.

19. Modify the backtracking program (*FindingPath*) which finds a path between a pair of nodes in a directed graph replacing the use of graph adjacency matrix by a compressed description of the graph (as in *CyclicGraphs2*).

20. Write a Fortran backtracking program to solve Sudoku puzzles (and e-mail the program to me when it is working; please **do not send** incorrect programs – I have more than enough of them).