**Ada Fundamentals with GNAT: Workshop 1**

*Note: where relevant, source programs are located in a workshop directory. The instructor will explain where these are found on the lab machines.*

1. Bug correction

Objective: This example illustrates some simple Ada features and gives practice in using the GNAT environment.

The program hello.adb in the workshop directory has an error. Make the necessary corrections and produce a running program.

2. Prime numbers

Objective: This problem illustrates basic Ada program structure and simple statements.

Write a procedure Display\_Primes that prompts the user for an integer N and then displays a table of all prime numbers less than or equal to N. (You may find it easiest to adapt the Display\_Squares procedure. Source files for this procedure and the Is\_Prime function are in the workshop directory.)

3. Car Talk Puzzler

Objective: This problem illustrates basic Ada program structure, floating point I/O, and simple statements.

a) One of the NPR Car Talk shows posed the following puzzler. A driver needs to go 75 miles. He goes at a speed of 75 mph for the first mile, then (with 74 miles remaining) he goes at 74 mph for the next mile, 73 mph for the 3rd mile, and so on, finishing up at 1 mph for the final mile. Calculate the time it takes for him to travel the full 75 miles, and display the result.

b) (Optional) As a generalization, assume that the driver needs to go a distance of D miles. He starts at a rate of D miles per hour, and then continuously adjusts his speed so that, for any value of d between 0.0 and D, his speed after traveling d miles is (D-d) miles per hour. First, compute the formula d(t), for the distance he has traveled at time t, as a function of t. Since this is a course in Ada and not algebra, if you need some guidance just look at the file puzzler-hint.txt in the workshop directory. The Ada part is to write a program that prompts for values of D and t and then displays the resulting value of d(t). (Although floating point computation needs to be done carefully in general, because of roundoff errors, do not worry about such subtleties in this example.)

4. Finite State Machine

Objective: This problem illustrates array handling and iteration.

Complete the implementation of the Odd\_Parity function shown on slide 78 in Part 1 of the lecture notes (Page 1/40 in the course notebook). Skeletal code and a driver procedure are supplied in the workshop directory.

5. Dice simulation

Objective: This problem illustrates array handling and simple statements

Write a program that prompts the user for an integer N and then simulates the throwing of a pair of dice N times. The workshop directory contains a package with the specification

**package** Random\_Numbers is **function** Random( K : Positive ) **return** Positive;  
**end** Random\_Numbers;

where the Random function returns a (pseudo) random number in the range 1..K. Your program should keep track of the number of times that each possible sum (2, 3, ..., 12) occurs. It should display the frequency for each of these values, and also the expected frequency based on the probability of each sum.

For those who need a reminder on probabilities, the following table shows the probability of a given sum coming up on any particular throw

Sum Probability

2 1/36  
3 2/36  
4 3/36  
5 4/36  
6 5/36  
7 6/36  
8 5/36  
9 4/36  
10 3/36  
11 2/36  
12 1/36

6. Checksum

Objective: This problem illustrates the relationship between enumeration types and integers.

The workshop directory contains a main program, Checksum\_Test, which calls a function with the specification:

**function** Checksum( Item : String ) **return** Natural;

The purpose of this function is to compute the “sum” of the characters in Item, where each character is treated as an integer value given by the character’s position in the Character type. Implement this function and test it; for example, the value displayed for the string ABC should be 198 (65+66+67).

7. Character count

Objective: This problem illustrates dealing with String data.

Write a program that accepts lines of text from the user, with an empty line terminating the input, and which displays the number of occurrences of each character that appeared in the input text. For example if the user enters the lines:

*Tried to run. Tried to hide.  
Break on through to the other side.*

your program should produce the following output:

Character Frequency  
' ' 12  
'.' 3  
'B' 1  
'T' 2  
'a' 1  
'd' 4  
'e' 7  
'g' 1  
'h' 5  
'i' 4  
'k' 1  
'n' 2  
'o' 6  
'r' 6  
's' 1  
't' 6  
'u' 2

Note: You may assume that the characters coming in are all graphic and hence may be displayed by calling the procedure Ada.Text\_IO.Put(Char). For simplicity output the characters in collating sequence order.

*If you’re feeling ambitious:* do not assume that the characters are all graphic. (For example, input may be supplied from a file rather than the keyboard). For a non-graphic character, display its position number instead of a character literal. The function Ada.Characters.Handling.Is\_Graphic(Char) takes a Character value Char and returns a Boolean indicating if Char is graphic. You need to “with” Ada.Characters.Handling to get access to this function. See [Ada Reference Manual §A.3.2].