Western Washington University
Computer Science Department

CSCI 141 Computer Programming I
Fall 2011

Laboratory Exercise 5

Objectives

1. Practice in using floating-point numbers.

2. Practice in exception handling.

Submitting Your Work

In this assignment you will write an Ada program to calculate statistical properties of some audio data. Save your program file (the .adb file) in the zipped tar file WnnnnnnnnLab5.tar.gz (where Wnnnnnnnn is your WWU W-number) and submit the file via the **Lab Exercise 5 Submission** item on the course web site. You must submit your program by 4:00pm on Friday, November 4.

The Audio Data

Several files are provided containing data from audio sources. A sound is converted into an electrical signal by a microphone. The electrical signal can then be converted into a series of numbers that represent the amplitudes of the electrical signal values over a brief period of time. It is these signal amplitude values that are recorded in the provided files.

Each file of audio data provided for this assignment is from recordings of a human voice saying a single word. Each data value is written as a real number in scientific notation, with a leading minus sign for negative values, a mantissa and an exponent to base 10. Here is a sample of the data:

```
-2.9296875e-03
-1.4648438e-03
-4.8828125e-04
 4.8828125e-04
 9.7656250e-04
 0.0000000e+00
 1.9531250e-03
```

Statistical Analysis

You are to calculate the following statistical properties of the data values. In the description below,

- d_i represents the ith data value
- $\sum_{i=1}^{n} d_i$ is the sum of the values $d_1, d_2, \ldots, d_n$

1. Number of data values in a file: n

2. Mean (average) of the data values: $\mu = \frac{1}{n}\sum_{i=1}^{n} d_i$

3. Variance of the data values: $\nu^2 = \frac{1}{n}\sum_{i=1}^{n} d_i^2 - \mu^2$

4. Average power: $\frac{1}{n}\sum_{i=1}^{n} d_i^2$

5. Average magnitude: $\frac{1}{n}\sum_{i=1}^{n} |d_i|$

6. Number of zero crossings (the number of time there is a change of sign between successive values).

What you need to do

1. You must declare a floating-point data type, with sufficient digits of precision for the data.

2. You must instantiate the Float_IO package to produce an IO package for your floating-point data type.

3. Since there is a large number of data values in each data file, your program cannot store them all in memory before you start the statistical analysis. You need to process each data value as it is input.

4. Your program will need to accumulate totals as it reads in the data. You will need totals for the sum of the data values, the sum of the squares of the data values, and the sum of the absolute values of the data values. You need to initialize these totals to zero before you start to read any of the data.

5. Your program will also need to count the number of data values and the number of times the sign changes between successive values (the zero crossings). These counts must also be set to zero before you start to read any data.

6. In order to detect zero crossings, you need to keep track of the previous data value so that you can compare its sign to the current data value. After you have done that comparison, you can set the previous value to the current value, in readiness for processing the next data value.

7. There may be errors in the data. Your program must use exception handling to report the error as (for example):

   ```
   Data item 23 is invalid
   ```

8. Following the detection and reporting of a data error, your program must continue processing with the next data item, omitting the invalid data from the statistical calculations.

9. After processing all the data in a file, your program must produce a neatly formatted display of the 6 statistical properties.

Submitting your program

1. Be sure that your program meets the coding standards listed below.

2. Include your program file in a zipped tar file (see previous labs for instructions).

Coding Standards

1. Use meaningful names that give the reader a clue as to the purpose of the thing being named.

2. Use comments at the start of the program to identify the purpose of the program, the author and the date written.

3. Use comments at the start of each procedure to describe the purpose of the procedure and the purpose of each parameter to the procedure.

4. Use comments at the start of each section of the program to explain what that part of the program does.

5. Use consistent indentation:

   - The declarations within a procedure must be indented from the Procedure and begin reserved words. The body of the procedure must be indented from the begin and end reserved words. Example of procedure indentation:

```
procedure DoStuff is
     Count : integer;
     Valid : boolean;
begin
     Count := 0;
     Valid := false;
end DoStuff;
```

   - The statements within the then-part, each elsif-part and the else-part of an if statement must be indented from the reserved words if, elsif, else and end.

```
if Count > 4 and not Valid then
     Result := 0;
     Valid := true:
elsif Count > 0 then
     result := 4;
else
     Valid := false;
end if;
```

   - The statements within a loop must be indented from the loop and end loop.

```
loop
     Count := Count + 1;
     Get (Number);
     exit when Number < Count;
end loop;
```

   - The exception handlers and statements within each exception handler must be indented.

```
begin
     ...  -- normal processing statements
exception
     when Exception1 =>
          Put_Line ("An error has occurred");
          Total := 0;
     when others =>
          Put_Line ("Something weird happened");
end;
```