Western Washington University
Computer Science Department

CSCI 141 Computer Programming I
Fall 2011

Laboratory Exercise 4

Objectives

1. Practice in writing procedures and functions and the use of parameters of various modes.

2. Practice in the use of enumerated types and subtypes.

3. Practice in producing zipped tar files in Linux.

Submitting Your Work

In this lab exercise you will develop an Ada program to display numbers in the style of a digital clock display. Save your program file (the .adb file) in the zipped tar file WnnnnnnnnLab4.tar.gz (where Wnnnnnnnn is your WWU W-number) and submit the file via the **Lab Exercise 4 Submission** item on the course web site. You must submit your program by 4:00pm on Friday, October 28.

The Exercise

In this lab exercise you will develop an Ada program to read in an integer number and display it in the style of a digital display, using only the space, underscore and vertical bar characters. For example, the numbers 10245 and 36978 would be displayed as:

```
    _     _        _           _    _    _   _    _
  |  |  |   _|  |_|  |_        _|  |_   |_|   |  |_|
  |  |_|  |_      |   _|       _|  |_|   _|   |  |_|
```

In optional extensions to this exercise, you can extend the program to:

1. Handle negative integers

2. Enable scaling of the display.

This lab exercise is to be completed in the Linux operating system. If your computer is currently running Windows, restart it in Ubuntu Linux.

Solving the Problem

The first step toward solving this problem is to recognize that the digital display of each digit has nine segments arranged in three horizontal zones, as shown below.

Top zone	Top left always a space	Top middle either space or '_'	Top right always a space
Upper zone	Upper left either space or '|'	Upper middle either space or '_'	Upper right either space or '|'
Lower zone	Lower left either space or '|'	Lower middle either 'space or '_'	Lower right either space or '|'

To produce the digital display for an integer number, we must follow these steps:

1. Prompt for and input the integer number to be displayed.

2. Ensure that it is a non-negative number. If a negative number is input, we can just change its sign – don't display an error message! The correct display of negative numbers is covered in an optional extension to this exercise.

3. For each zone (top, upper and lower), display that part of all the digits in the number and then skip to a new line.

To display a zone of all the digits in a number, use a procedure which has in-mode parameters for the number to be displayed and the zone of the number to be displayed.

Use an enumerated type for the zones:

```
type zone is (top, upper, lower);
```

Then step 3 above can be handled by a for-loop:

```
for thisZone in zone loop
     Display (Number, thisZone);
end loop;
```

The loop variable `thisZone` is implicitly declared with type `zone` within the loop but can be used only within the loop.

Since the number passed to procedure Display is nonnegative, we can make the parameter type `natural`.

The heading for the procedure should then look like:

```
procedure Display ( Number : in natural;
                    Part   : in zone      ) is
```

Within procedure Display, we need to display the characters of the specified zone for each digit in the number. There are two initial problems here:

1. How many digits are there in the number?
2. How do we isolate each digit?

To solve the first problem, we use a function `DigitCount` which takes the number as a parameter and returns the number of digits:

```
function DigitCount (Number : in natural)
                     return natural is
```

The method to use for this function is simply to keep dividing the number by 10 until you get to a number which is less than 10 (only one digit). Count the number of times you divide by 10 and the number of digits in the number is that count plus 1. For example, if the number is 4326, the values in the function's iterations are:

Number	Count
4326	0
432	1
43	2
4	3

So, the number of digits in the number is the final value of Count + 1, which equals 4.

To isolate the nth digit of the number, we use a function NthDigit, which takes the number and the digit position as parameters and returns a digit.

For the return type, define a subtype Digit.

```
subtype Digit is integer range 0 .. 9;
```

Then the function heading can be:

```
function NthDigit (Number, Position : in natural)
                   return Digit is
```

The body for this function makes use of modulo division and integer division. It helps if we number the digits from the right, starting with 1. This may seem counter-intuitive, but it makes the calculation much easier than if we number the digits from the left. So, for the number 4326, the 1st digit is 6, the 2nd digit is 2, the 3rd digit is 3 and the 4th digit is 4. The calculations are shown below.

n	Calculation	Result
1	4326 mod 10 / 1 = 6 / 1	6
2	(4326 mod 100) / 10 = 26 / 10	2
3	(4326 mod 1000) / 100 = 326 / 100	3
4	(4326 mod 10000) / 1000 = 4326 / 1000	4

Now within procedure Display, we can go through the digits in reverse order (the leftmost digit first) and output the zone characters of that digit.

```
for n in reverse 1 .. DigitCount (Number) loop
     Output ( NthDigit(Number, n), Part);
end loop;
```

The heading for procedure OutPut is:

```
procedure Output ( thisDigit : in Digit;
                   thisZone  : in Zone   ) is
```

Procedure Output needs to output the three characters for the digit specified by `thisDigit`, for the zone specified by `thisZone`.

In order to tell procedure Output which character (space, '_' or '|') to display for each segment of a digit, write a function which is given the digit as a parameter and returns a character. Since there are 9 segments and the top-left and top-right segments of every digit are always spaces, that leaves 7 functions to write. Here is function TopMiddle as an example.

```
function TopMiddle ( thisDigit : in Digit )
                     return character is
begin
     case thisDigit is
     when 0 | 2 | 3| 5 | 6 | 7 | 8 | 9 => return '_';
     when 1 | 4                        => return ' ';
     end case;
end TopMiddle;
```

The remaining functions are UpperLeft, UpperMiddle, UpperRight, LowerLeft, LowerMiddle and LowerRight. You get to write those yourself!

Then, within procedure Output, if `thisZone` is top, it displays the `top` zone of the digit by the following calls to Ada.Text_IO.Put:

```
Put (' ');
Put (TopMiddle (thisDigit));
Put (' ');
```

There will be a sequence of 3 Put calls for the upper zone and a sequence of 3 Put calls for the lower zone. You get to figure those out.

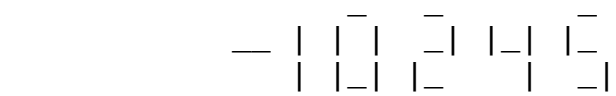
You should leave an extra space after the zone output for the digit to ensure that there is a space between digits in the digital display.

Bonus Sections

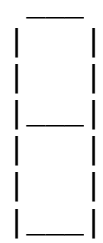
A working program, constructed in the way described above, using the coding standards specified on the next page is sufficient for full points on this lab exercise.

There are two extra credit extensions to the exercise.

1. (2 bonus points) Enable the display of negative numbers, using a leading minus sign. So, for example, the number -10245 would display as follows (using two underscore characters for the minus sign).



2. (3 bonus points) Enable the numbers to be displayed in scaled form, with the scale specified by the user of the program. In this case, after prompting for and reading the number to be displayed, the program must prompt for and read the scale to be used for the display. The display produced for the program so far is with scale equal to 1. A scale of 2 must double the height and width of the digits, a scale of 3 must triple their size, and so on up to a scale of 5. For example, the digit 8, displayed with scale 3 is



When displaying scaled negative numbers, don't scale the leading minus sign (it looks stupid!).

Submitting your program

1. Be sure that your program meets the coding standards listed on the next page.

2. Include your program file in a zipped tar file (see Lab 2 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;
```