Western Washington University
Computer Science Department

CSCI 145 Computer Programming and Linear Data Structures
Winter 2012

Assignment 2

Submitting Your Work

This assignment is worth 15% of the grade for the course. In this assignment you will create the specification and body of an Ada package for a string type lstring implemented as a linked list of short, fixed-length strings. Save your package files (the .ads and .adb file) in the zipped tar file WnnnnnnnnAssg2.tar.gz (where Wnnnnnnnn is your WWU W-number) and submit the file via the **Assignment 2 Submission** item on the course web site. You must submit your assignment by 5:00pm on Friday, February 24.

Linked Strings

Ada's existing String type is a fixed-length array of characters. When a variable of type string is assigned a value, that value must have exactly the same length as the variable's string type. Variables of type string cannot be extended beyond their declared length. The Ada string type is very restrictive and can be quite frustrating to work with.

The linked string is an alternative way to store string values. Instead of using just one array of characters of exactly the correct length, the linked string uses a linked list of short, fixed-length strings. Linked strings can be of any length and their length can change during program execution.

However, linked strings still need the convenient operations of Ada's string type. In addition, there must be functions to convert a string to an lstring, and convert an lstring to a string.

The lstring type must be limited private, meaning that there is no access to the components of a lstring except through the procedures and functions defined in the linked string package. Furthermore, the assign operator := and the equality relational operators = and /= are not defined for the lstring data type.

The procedures and functions required for the lstring type are listed below.

Function toLstring	Given a string, creates and returns an lstring with the same contents as the string.
Function toString	Given an lstring, creates and returns a string with the same contents as the lstring.

Procedure Get	Read an lstring as the entire line from standard input (without the end of line character).
Procedure Get	Given a parameter of type Ada.Text_IO.File_type, read an lstring as an entire line from the file represented by that File_type parameter (without the end of line character).
Procedure Put	Output an lstring to standard output.
Procedure Put	Given a parameter of type Ada.Text_IO.File_type, output an lstring to the file represented by that File_type parameter
Procedure Put_Line	Output an lstring to standard output, followed by a new line.
Procedure Put_Line	Given a parameter of type Ada.Text_IO.File_type, output an lstring to the file represented by that File_type parameter, followed by a new line.
Function "&"	A binary operator to concantenate two lstring values, producing a third lstring value.
Procedure Copy	This replaces the := operator for the limited private type. Given a source lstring, copies that lstring to a target lstring.
Function Slice	Given an lstring and start and finish positions in that lstring, return an lstring which is a slice of the original lstring. For example, if lstr is an lstring variable, Slice(lstr, 5, 12) is an lstring containing the 5th through 12th characters of lstr.
Function "="	A binary operator to compare two lstring values for equality.
Function "<"	A binary operator to compare two lstring values and return true if the first lstring would come before the second lstring value, in dictionary order, and return false otherwise.
Function ">"	A binary operator to compare two lstring values and return true if the first lstring would come after the second lstring value, in dictionary order, and return false otherwise.
Function Length	Return the length (number of characters) in an lstring.

Program Requirements

1. Your package must provide the limited private data type lstring, implemented as a linked list of short, fixed-length strings.

2. Your package must provide the procedures and functions listed above for creation, I/O, manipulation and determining properties of lstring values.

3. Your package must enable the provided test program string_test.adb to be compiled and run correctly.

What you must submit

You must submit the Ada source files for the linked string package (the .ads and .adb file). These files must be included in a zipped tar file.

Saving your files in a zipped tar file

You only submit .adb and .ads files. First you need to bundle them up into a single tar file.

Use the command:

```
tar -cf WnnnnnnnnAssg2.tar *.adb *.ads
```

(where Wnnnnnnnn is your W-number).

The -cf specifies two options for the tar command: 'c' means create and 'f' means that the name of the resulting tar file comes next in the command.

This will include every file in the current directory whose name ends with ".adb" or ".ads".

If you now use the command `ls` you should now see the file WnnnnnnnnAssg2.tar in your directory.

If you use the command

```
tar -tf WnnnnnnnnAssg2.tar
```

(where Wnnnnnnnn is your W-number), it will list the files within the tar file.

Now compress the tar file using the gzip program:

```
gzip WnnnnnnnnAssg2.tar
```

By using the `ls` command again, you should see the file WnnnnnnnnAssg2.tar.gz in your directory. This is the file that you need to submit through the **Assignment 2 Submission** link in the moodle web site.

Coding Standards

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

2. Use named constants instead of repeated use of the same numeric constant.

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

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

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

6. 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;
```