Programación de Sistemas de Telecomunicación / Informática II
Práctica 1:
Trocear frases en palabras y contarlas

Departamento de Sistemas Telemáticos y Computación
(GSyC)

Septiembre de 2013

1. Introducción

En esta práctica debes realizar dos programas en Ada relacionados con la gestión de palabras y caracteres de un texto.

El primer programa (`trocea`) pedirá al usuario una línea de caracteres y mostrará en pantalla una a una las palabras que contiene, detallando al final el total de palabras y espacios en blanco.

El segundo programa (`cuenta`) analizará las líneas de un fichero de texto mostrando en pantalla el total de líneas, palabras y caracteres que contiene, y mostrará el número de veces que aparece en el fichero cada palabra.

2. Descripción del programa `trocea`

Escribe en lenguaje Ada un programa llamado `trocea` que pida al usuario una cadena de caracteres, y que muestre cada una de sus palabras en una línea distinta. El programa debe reconocer como palabras cualquier conjunto de caracteres separado de otros por **uno o más espacios en blanco seguidos**.

Al final, el programa escribirá el número de palabras reconocidas y el número de espacios en blanco.

Ejemplos de ejecución:

```
$ ./trocea
Introduce una cadena: Esto es un ejemplo
Palabra 1: |Esto|
Palabra 2: |es|
Palabra 3: |un|
Palabra 4: |ejemplo|
Total: 4 palabras y 3 espacios.

$ ./trocea
Introduce una cadena:      Esto      es    un    ejemplo
Palabra 1: |Esto|
Palabra 2: |es|
Palabra 3: |un|
Palabra 4: |ejemplo|
Total: 4 palabras y 23 espacios.

$ ./trocea
```

```
Introduce una cadena: Esto es, un ejemplo.
Palabra 1: |Esto|
Palabra 2: |es,|
Palabra 3: |un|
Palabra 4: |ejemplo.|
Total: 4 palabras y 3 espacios.

$ ./trocea
Introduce una cadena:    Esto
Palabra 1: |Esto|
Total: 1 palabra y 4 espacios.

$ ./trocea
Introduce una cadena: 1
Palabra 1: |1|
Total: 1 palabra y 0 espacios.
```

3. Descripción del programa `cuenta`

Escribe en lenguaje Ada un programa llamado `cuenta` que cuente el número de líneas, palabras y caracteres que contiene un fichero de texto cuyo nombre se le pasa como argumento en la línea de comandos.

Al ejecutar el programa se le pasarán obligatoriamente los siguientes dos argumentos: `-f` y a continuación el nombre del fichero que contiene las palabras a contar.

Adicionalmente se le podrá pasar al programa el argumento `-t`, antes o después de los 2 argumentos obligatorios. Si el argumento `-t` está presente el programa deberá mostrar, después de la información del total de líneas, palabras y caracteres del fichero, una tabla con el número de veces que aparece cada una de las palabras del fichero. Si el argumento `-t` no está presente, la tabla de palabras no aparecerá.

Notas:

- Se deberá reconocer como palabras cualquier conjunto de caracteres separado de otros por **uno o más espacios en blanco seguidos**.
- Se deberán reconocer como caracteres cualquier carácter contenido en el fichero, incluyendo los espacios en blanco o el carácter de fin de línea.
- A la hora de diferenciar palabras NO se tendrán en cuenta minúsculas y mayúsculas, por lo que `"hola"`, `"Hola"` y `"HOLA"` se considerarán **la misma palabra**.
- A la hora de pasarle argumentos al programa, son correctas las formas siguientes:
  - `./cuenta -f fichero.txt`
  - `./cuenta -t -f fichero.txt`
  - `./cuenta -f fichero.txt -t`

  Y son incorrectas todas las demás formas.

Ejemplos de ejecución (para cada fichero, primero se muesta con `cat` su contenido, y luego se ejecuta el programa):

```
$ cat f1.txt
hola

$ ./cuenta -f f1.txt
2 líneas, 1 palabra, 6 caracteres

$ cat f2.txt
hola a todos
otra vez
hola a todos

$ ./cuenta -f f2.txt
4 líneas, 8 palabras, 36 caracteres

$ cat f3.txt
hola    a    todos


   otra    vez

$ ./cuenta -f f3.txt
4 líneas, 5 palabras, 36 caracteres

$ ./cuenta -f f2.txt -t
4 líneas, 8 palabras, 36 caracteres


Palabras
--------
hola: 2
a: 2
todos: 2
otra: 1
vez: 1

$ ./cuenta -t -f f3.txt
4 líneas, 5 palabras, 36 caracteres


Palabras
--------
hola: 1
a: 1
todos: 1
otra: 1
vez: 1
```

Para saber si tu programa `cuenta` está funcionando bien, compara su salida con la que muestra la orden de la shell `wc` (*word count*):

```
$ wc f1.txt
 2  1 6 f1.txt

$ wc f2.txt
 4  8 36 f2.txt

$ wc f3.txt
 4  5 36 f3.txt
```

4. Pautas de Implementación

1. Para no repetir código entre los dos programas, se aconseja crear un paquete auxiliar que incluya los subprogramas necesarios para contar las palabras de una cadena de caracteres, y utilizar dicho paquete desde los dos programas principales `trocea.adb` y `cuenta.adb`.

2. Los programas deberán escribirse teniendo en cuenta las consideraciones sobre legibilidad del código que hemos comentado en clase.

3. Los programas deberán ser robustos, comportándose de manera adecuada cuando no se arranquen con los parámetros en línea de comandos adecuados.

4. Para almacenar las palabras que van apareciendo de forma que se puedan contabilizar sus apariciones en el fichero se utilizará la definición del tipo `Cell` que aparece en el paquete `List`, cuyo contenido incompleto de la especificación se muestra a continuación. El alumno deberá completar el paquete y hacer uso de él para implementar el programa.

   ```
   with Ada.Strings.Unbounded;
   package List is

      package ASU renames Ada.Strings.Unbounded;

      type Cell;
      type Cell_A is access Cell;

      type Cell is record
         Name : ASU.Unbounded_String;
         Count: Natural := 0;
         Next : Cell_A;
      end record;

   end List;
   ```

   Antes de terminar de terminar el programa debe liberarse la memoria ocupada por la lista de palabras.

5. Consulta el código del Ejemplo Final[1] de las transparencias de Introducción a Ada, y en particular estudia cómo se utilizan los subprogramas `Length`, `Index`, `Head` y `Tail` del paquete `Ada.Strings.Unbounded`. Estos subprogramas se explican más adelante (ver apartado 5).

   Puedes reutilizar código del mencionado Ejemplo Final, pero ten en cuenta que el procedimiento `Next_Token` que aparece en dicho Ejemplo Final no funcionará directamente en esta práctica, sino que será necesario realizarle algunas modificaciones.

6. Para la gestión de ficheros de texto consulta las transparencias de Introducción a Ada y el ejemplo que aparece más adelante (ver apartado 6).

5. Subprogramas para el manejo de `Unbounded_Strings`

Todos los subprogramas que se detallan a continuación pertenecen al paquete `Ada.Strings.Unbounded`.

- Función `Length`: Función devuelve la longitud (número de caracteres) del `Unbounded_String` que recibe como parámetro.

```
   function Length
     (Source  : Unbounded_String) return Natural;
```

  - `Source`: `Unbounded_String` cuya longitud quiere conocerse.
  - Devuelve un `Natural` indicando la longitud de `Source`.

  Ejemplo de uso:

```
   S: ASU.Unbounded_String := ASU.To_Unbounded_String("Hola a todos");
   N: Natural;
...
   N := ASU.Length (S);     -- N tomará el valor 12
```

- Función `Index`: Función que busca dentro de un `Unbounded_String` una subcadena, devolviendo la posición en la que aparece.

```
   function Index
     (Source  : Unbounded_String;
      Pattern : String;
      Going   : Direction := Forward;
      Mapping : Maps.Character_Mapping := Maps.Identity) return Natural;
```

  - `Source`: `Unbounded_String` en el que se busca.
  - `Pattern`: `String` que contiene la cadena de caracteres que se busca dentro de `Source`.
  - `Going`: Dirección en la que se busca, por defecto hacia adelante. Como este parámetro tiene un valor por defecto no es necesario pasarlo en la llamada.
  - `Mapping`: Correspondencia entre juegos de caracteres, por defecto la identidad. Como este parámetro tiene un valor por defecto no es necesario pasarlo en la llamada.

[1] `./ip_puerto.adb`

- Devuelve un `Natural` indicando la posición de `Source` en la que aparece por primera vez el patrón `Pattern` buscado. **Si el patrón de búsqueda no se encuentra, se devuelve el valor `0`.**

Ejemplo de uso:

```
   S: ASU.Unbounded_String := ASU.To_Unbounded_String("Hola a todos");
   N: Natural;
...
   N := ASU.Index (S, "to");     -- N tomará el valor 8
```

- Función `Head`: Función que devuelve un trozo del principio de un `Unbounded_String`.

```
   function Head
     (Source : Unbounded_String;
      Count  : Natural;
      Pad    : Character := Space) return Unbounded_String;
```

  - `Source`: `Unbounded_String` del que se quiere extraer un trozo del principio.
  - `Count`: Número de caracteres de `Source` a devolver.
  - `Pad`: Carácter de relleno, por defecto un espacio en blanco. Como este parámetro tiene un valor por defecto no es necesario pasarlo en la llamada.
  - Devuelve un `Unbounded_String` con los `Count` primeros caracteres de `Source`. Si `Source` tuviera menos de `Count` caracteres, el valor devuelto se completa con caracteres iguales a `Pad`.

Ejemplo de uso:

```
   S: ASU.Unbounded_String := ASU.To_Unbounded_String("Hola a todos");
   R: ASU.Unbounded_String;
   N: Natural;
...
   N := ASU.Index (S, "to");     -- N tomará el valor 8
   R := ASU.Head (S, N-1);       -- R almacenará la cadena "Hola a "
```

- Procedimiento `Head`: Procedimiento que trunca un `Unbounded_String` para quedarse con el principio.

```
   procedure Head
     (Source : in out Unbounded_String;
      Count  : Natural;
      Pad    : Character := Space);
```

  - `Source`: `Unbounded_String` del que se quiere extraer un trozo del principio. El resultado se deja en este mismo parámetro, que se recibe en modo `in out`.
  - `Count`: Número de caracteres de `Source` que se quieren conservar.
  - `Pad`: Carácter de relleno, por defecto un espacio en blanco. Como este parámetro tiene un valor por defecto no es necesario pasarlo en la llamada. Si `Source` tuviera menos de `Count` caracteres, en vez de truncarse se completa con caracteres iguales a éste.

Ejemplo de uso:

```
   S: ASU.Unbounded_String := ASU.To_Unbounded_String("Hola a todos");
   N: Natural;
...
   N := ASU.Index (S, "to");     -- N tomará el valor 8
   ASU.Head (S, N-1);            -- S almacenará la cadena "Hola a "
```

- Función `Tail`: Función que devuelve un trozo del final de un `Unbounded_String`.

```
   function Tail
     (Source : Unbounded_String;
      Count  : Natural;
      Pad    : Character := Space) return Unbounded_String;
```

  - `Source`: `Unbounded_String` del que se quiere extraer un trozo del final.
  - `Count`: Número de caracteres de `Source` a devolver.
  - `Pad`: Carácter de relleno, por defecto un espacio en blanco. Como este parámetro tiene un valor por defecto no es necesario pasarlo en la llamada.
  - Devuelve un `Unbounded_String` con los `Count` últimos caracteres de `Source`. Si `Source` tuviera menos de `Count` caracteres, el valor devuelto se completa con caracteres iguales a `Pad`.

Ejemplo de uso:

```
   S: ASU.Unbounded_String := ASU.To_Unbounded_String("Hola a todos");
   R: ASU.Unbounded_String;
   N: Natural;
...
   N := ASU.Index (S, "to");              -- N tomará el valor 8
   R := ASU.Tail (S, ASU.Length(S)-N+1); -- R almacenará la cadena "todos"
```

- Procedimiento `Tail`: Procedimiento que trunca un `Unbounded_String` para quedarse con el final.

```
   procedure Tail
     (Source : in out Unbounded_String;
      Count  : Natural;
      Pad    : Character := Space);
```

  - `Source`: `Unbounded_String` del que se quiere extraer un trozo del final. El resultado se deja en este mismo parámetro, que se recibe en modo `in out`.
  - `Count`: Número de caracteres de `Source` que se quieren conservar.
  - `Pad`: Carácter de relleno, por defecto un espacio en blanco. Como este parámetro tiene un valor por defecto no es necesario pasarlo en la llamada. Si `Source` tuviera menos de `Count` caracteres, en vez de truncarse se completa con caracteres iguales a éste.

Ejemplo de uso:

```
   S: ASU.Unbounded_String := ASU.To_Unbounded_String("Hola a todos");
   N: Natural;
...
   N := ASU.Index (S, "to");         -- N tomará el valor 8
   ASU.Tail (S, ASU.Length(S)-N+1); -- S almacenará la cadena "todos"
```

Puedes consultar el Manual de Referencia de Ada para ver la especificación completa de estos y otros subprogramas para el tratamiento de cadenas de caracteres.

6. Gestión de ficheros de texto

El paquete `Ada.Text_IO` permite, además de la entrada y salida de texto a través de la entrada y salida estándar (por defecto, teclado y ventana de terminal), leer y escribir en ficheros de texto utilizando los mismos subprogramas pero utilizando un primer parámetro adicional que representa al fichero del que se lee y se escribe.

El siguiente programa es un ejemplo simple de un programa que copia el contenido de un fichero de texto en otro:

```
with Ada.Text_IO;
with Ada.Strings.Unbounded;
with Ada.Command_Line;
with Ada.IO_Exceptions;


procedure Copia is
   package ASU renames Ada.Strings.Unbounded;
   package T_IO renames Ada.Text_IO;

   Usage_Error: exception;

   Fichero_Origen: T_IO.File_Type;
   Fichero_Destino: T_IO.File_Type;

   S: ASU.Unbounded_String;
   Terminar: Boolean;

begin

   if Ada.Command_Line.Argument_Count /= 2 then
      raise Usage_Error;
   end if;

   T_IO.Open(Fichero_Origen, T_IO.In_File, Ada.Command_Line.Argument(1));
   T_IO.Create(Fichero_Destino, T_IO.Out_File, Ada.Command_Line.Argument(2));

   Terminar := False;
   while not Terminar loop
      begin
         S := ASU.To_Unbounded_String(T_IO.Get_Line(Fichero_Origen));
         T_IO.Put_Line(Fichero_Destino, ASU.To_String(S));
      exception
         when Ada.IO_Exceptions.End_Error =>
            Terminar := True;
      end;
   end loop;

   T_IO.Close(Fichero_Origen);
   T_IO.Close(Fichero_Destino);

exception
   when Usage_Error =>
      T_IO.Put_Line("uso: copia <fichero-origen> <fichero-destino>");

end Copia;
```

7. Normas de entrega

7.1. Alumnos de Programación de Sistemas de Telecomunicación

Debes dejar el código fuente de tu programa dentro de tu cuenta en los laboratorios de prácticas. Directamente en el directorio de tu cuenta debes crear una carpeta de nombre `PST.2013`, y dentro de ella una carpeta de nombre `P1`. En ella deben estar todos los ficheros correspondientes a los dos programas (`trocea.adb`, `cuenta.adb`), y a la lista (`list.ads`, `list.adb`), así como otros ficheros de paquetes que pudieras haber creado adicionalmente.
Importante: Pon un comentario con tu nombre completo en la primera línea de cada fichero fuente.

7.2. Alumnos de Informática II

Debes dejar el código fuente de tu programa dentro de tu cuenta en los laboratorios de prácticas. Directamente en el directorio de tu cuenta debes crear una carpeta de nombre `I2.2013`, y dentro de ella una carpeta de nombre `P1`. En ella deben estar todos los ficheros correspondientes a los dos programas (`trocea.adb`, `cuenta.adb`), y a la lista (`list.ads`, `list.adb`), así como otros ficheros de paquetes que pudieras haber creado adicionalmente.
Importante: Pon un comentario con tu nombre completo en la primera línea de cada fichero fuente.

8. Plazo de entrega

Los ficheros deben estar en tu cuenta, en la carpeta especificada más arriba, antes de las 23:59h del viernes 11 de octubre de 2013.