Programación de Sistemas de Telecomunicación / Informática II
Práctica 1:
Contar palabras

GSyC

Septiembre de 2015

1. Introducción

En esta práctica debes realizar un programa en Ada relacionado con la gestión de palabras de un fichero de texto.

El programa analizará las líneas del fichero y almacenará en una lista dinámica las palabras diferentes que contiene, incluyendo el número de veces que aparece cada una. Además el programa permitirá añadir, borrar, o buscar palabras a la lista, y al terminar mostrará la palabra más repetida de la lista.

2. Especificación del programa `words`

Escribe en lenguaje Ada un programa llamado `words` que almacene una lista con las palabras que contiene un fichero de texto y la frecuencia de aparición de cada una de ellas (es decir, cuántas veces aparece cada palabra en el fichero de texto).

Al ejecutar el programa se le pasará obligatoriamente como argumento el nombre del fichero que contiene las palabras a contar. Y el programa tras construir la lista de palabras mostrará por pantalla la palabra que más veces aparece en el fichero.

Adicionalmente se le podrán pasar (o no) al programa dos argumentos más:

`-l` Si se incluye este argumento, el programa, tras analizar el fichero, mostrará la lista de todas las palabras diferentes que contenía, y la frecuencia de apariciones de cada una.

`-i` Si se incluye este argumento, el programa, tras analizar el fichero, funcionará de forma interactiva, presentando al usuario un menú con las siguientes opciones:

1. Añadir una palabra a la lista: Si la palabra a añadir ya está en la lista, se incrementará en uno su frecuencia. Si la palabra es nueva, se añadirá a la lista con una frecuencia de 1.
2. Borrar una palabra de la lista
3. Buscar una palabra en la lista, mostrando cuál es su frecuencia
4. Mostrar todas las palabras de la lista.
5. Salir del programa

Si aparecieran tanto las opciones `-l` como `-i`, el programa primero mostrará la lista de palabras, luego pasará a modo interactivo, y por último mostrará la palabra de mayor frecuencia de la lista final.

El programa, por lo tanto, debe poder lanzarse de las siguientes formas:

- `./words f1.txt`
  Construye la lista de palabras, pero sólo muestra la palabra más frecuente.
- `./words -l f1.txt`
  Construye la lista de palabras, muestra la lista de palabras con sus frecuencias, y termina mostrando la palabra más frecuente.
- `./words -i f1.txt`
  Construye la lista de palabras, y entra en modo interactivo. Cuando se elige la opción de salir (5), se muestra la palabra más frecuente y se termina la ejecución.

- `./words -l -i f1.txt`
  Construye la lista de palabras, muestra la lista de palabras con sus frecuencias, y entra en modo interactivo. Cuando se elige la opción de salir, se muestra la palabra más frecuente y se termina la ejecución.

- `./words -i -l f1.txt`
  Igual que en el caso anterior.

Cualquier otra forma de lanzar el programa debe hacer que éste termine sin hacer nada, indicando cuáles son la formas correctas de lanzarlo.

3. Ejemplos de ejecución

Antes de la primera ejecución se muestra con `cat` el contenido del fichero:

```
$ cat f1.txt
hola a todos
  hola  a  todos
hola
  a
todos
hola
mundo
adios



$ ./words f1.txt
The most frequent word: |hola| - 4



$ ./words -l f1.txt
|hola| - 4
|a| - 3
|todos| - 3
|mundo| - 1
|adios| - 1
The most frequent word: |hola| - 4



$ ./words -i f1.txt

Options
1 Add word
2 Delete word
3 Search word
4 Show all words
5 Quit

Your option? 4

|hola| - 4
|a| - 3
|todos| - 3
|mundo| - 1
|adios| - 1
```

```
Options
1 Add word
2 Delete word
3 Search word
4 Show all words
5 Quit

Your option? 1
Word? casa
Word |casa| added

Options
1 Add word
2 Delete word
3 Search word
4 Show all words
5 Quit

Your option? 4

|hola| - 4
|a| - 3
|todos| - 3
|mundo| - 1
|adios| - 1
|casa| - 1

Options
1 Add word
2 Delete word
3 Search word
4 Show all words
5 Quit

Your option? 2
Word? a

|a| deleted

Options
1 Add word
2 Delete word
3 Search word
4 Show all words
5 Quit

Your option? 4

|hola| - 4
|todos| - 3
|mundo| - 1
|adios| - 1
|casa| - 1

Options
1 Add word
```

```
2 Delete word
3 Search word
4 Show all words
5 Quit

Your option? 3
Word? todos

|todos| - 3

Options
1 Add word
2 Delete word
3 Search word
4 Show all words
5 Quit

Your option? 5

The most frequent word: |hola| - 4



$ ./words -l -i f1.txt
|hola| - 4
|a| - 3
|todos| - 3
|mundo| - 1
|adios| - 1

Options
1 Add word
2 Delete word
3 Search word
4 Show all words
5 Quit

Your option? 5

The most frequent word: |hola| - 4
```

4. Condiciones obligatorias de funcionamiento

1. Los programas deberán escribirse teniendo en cuenta las consideraciones sobre legibilidad y reutilización del código que hemos comentado en clase.
2. Los programas deberán ser robustos, comportándose de manera adecuada cuando no se arranquen con los parámetros adecuados en línea de comandos.
3. Las palabras deben almacenarse en una lista dinámica. Dicha lista dinámica debe estar implementada en el paquete `Word_Lists` cuya especificación (que no puede modificarse) es la siguiente:

```
with Ada.Strings.Unbounded;

package Word_Lists is
   package ASU renames Ada.Strings.Unbounded;

   type Cell;

   type Word_List_Type is access Cell;

   type Cell is record
      Word: ASU.Unbounded_String;
      Count: Natural := 0;
      Next: Word_List_Type;
   end record;

   Word_List_Error: exception;

   procedure Add_Word (List: in out Word_List_Type;
                       Word: in ASU.Unbounded_String);

   procedure Delete_Word (List: in out Word_List_Type;
                          Word: in ASU.Unbounded_String);

   procedure Search_Word (List: in Word_List_Type;
                          Word: in ASU.Unbounded_String;
                          Count: out Natural);

   procedure Max_Word (List: in Word_List_Type;
                       Word: out ASU.Unbounded_String;
                       Count: out Natural);

   procedure Print_All (List: in Word_List_Type);

end Word_Lists;
```

4. Comportamiento esperado de los subprogramas:
   - `Add_Word`: Si `Word` ya está en la lista, incrementa en un su `Count`. Si `Word` no está en la lista, crea una nueva celda para ella, con `Count` a 1.
   - `Delete_Word`: Si `Word` está en la lista, elimina su celda de la lista y libera la memoria ocupada por ella (llamando adecuadamente a `Free`). Si `Word` no está en la lista, eleva la excepción `Word_List_Error`.
   - `Search_Word`: Si `Word` está en la lista, devuelve su `Count`. Si `Word` no está en la lista, devuelve `0`.
   - `Max_Word`: Devuelve los campos de la celda de mayor `Count` de la lista. Si hay varias celdas con el mismo valor máximo de `Count`, devuelve la primera de ellas. Si la lista está vacía, eleva la excepción `Word_List_Error`.
   - `Print_All`: Muestra el contenido de todas las celdas de la lista, en el mismo orden en que se introdujeron en ella, y con el formato que se muestra en los ejemplos de ejecución. Si la lista está vacía, muestra el mensaje `No words`.

5. Cuando se muestre la lista de palabras, las palabras deben aparecer **en el mismo orden** en el que aparecen en el fichero de texto (como se muestra en los ejemplos de ejecución)
6. La salida del programa deberá ser **exactamente igual** a la que se muestra en los ejemplos de ejecución, con el mismo formato y la misma cadena de mensajes.
7. Se deberá reconocer como palabras cualquier conjunto de caracteres separado de otros por **uno o más espacios en blanco seguidos**.
8. Dos palabras sólo son iguales si todos sus caracteres son idénticos, incluyendo signos de puntuación y mayúsculas o minúsculas. Así, serán palabras distintas `casa`, `Casa`, `CaSa`, `casa,`, `casa.`, `-Casa`...
9. Por tanto, en la línea:

```
-Luis por fin no viene, me temo.
```

   las palabras son `|-Luis|`, `|por|`, `|fin|`, `|no|`, `|viene,|`, `|me|`, `|temo.|` (Nota: los caracteres `|` se usan sólo para mostrar mejor los límites de la palabra, pero realmente no son parte de la misma)
10. El programa se usará sólo con ficheros de texto con caracteres ASCII, como los que hemos proporcionado de ejemplo.

5. Extensiones

Una vez que la práctica funcione como se indica en el apartado anterior, puedes realizarle alguna de las siguientes extensiones. Ten en cuenta que en la entrega de la práctica pueden pedirse modificaciones o extensiones del tipo de las que aparecen en este apartado:

1. Se considerarán separadores de palabras uno o más caracteres seguidos no alfanuméricos. Por tanto, en la línea

```
-Luis por fin no viene, me temo.
```

   las palabras son `|Luis|`, `|por|`, `|fin|`, `|no|`, `|viene|`, `|me|`, `|temo|`
2. Se considerarán iguales palabras que sólo se diferencian en mayúsculas/minúsculas. Así, debe contarse como la misma palabra `casa`, `Casa`, `CASA`, `CaSa`...
3. Antes de terminar el programa debe liberarse la memoria ocupada por la lista de palabras, para lo cual podrás modificar la especificación del paquete `Word_Lists` para incluir el siguiente procedimiento:

```
procedure Delete_List (List: in out Word_List_Type);
```

   Al llamar a este procedimiento se irá borrando elemento a elemento de la lista, liberando la memoria asociada, hasta que la lista quede vacía.

6. Pautas de Implementación

1. Para extraer de cada línea del fichero de texto las palabras que contiene, utiliza los subprogramas `Index`, `Head` y `Tail` del paquete `Ada.Strings.Unbounded`, cuya funcionamiento se describe en el apartado 7. Para cada subprograma se incluye un ejemplo de uso que facilita mucho entender cómo funciona.
2. Puedes reutilizar código del procedimiento `Next_Token` que aparece en el Ejemplo Final de las transparencias de Introducción a Ada, pero ten en cuenta que dicho código no funcionará directamente en esta práctica, sino que será necesario realizarle algunas modificaciones.
3. Para la gestión de ficheros de texto os proporcionamos el ejemplo que aparece más adelante (ver apartado 8), que puede usarse tal cual.
4. Para la extensión 1 puedes usar la versión de la función `Index` que aparece al final de apartado 7. Esta función utiliza `Maps` para especificar distintos caracteres alternativos a buscar, en vez de buscar exactamente un carácter o una subcadena.
5. Para la extensión 2 puedes usar la función `Ada.Characters.Handling.To_Lower()` que recibe como único parámetro un `String` y devuelve un `String` resultado de pasar el parámetro a minúsculas.

7. Subprogramas para el manejo de `Unbounded_Strings`

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

- Función `Length`: Función que 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.
  - 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, " ");     -- N tomará el valor 5
```

- Función `Head`: Función que devuelve una parte de un `Unbounded_String`, comenzando por el principio de dicho `Unbounded_String`.

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

  - `Source`: `Unbounded_String` del que se quiere extraer una parte 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, " ");      -- N tomará el valor 5
   R := ASU.Head (S, N-1);       -- R almacenará la cadena "Hola"
```

- 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 una parte 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, el valor devuelto se completa con caracteres iguales a `Pad`.

  Ejemplo de uso:

```
   S: ASU.Unbounded_String := ASU.To_Unbounded_String("Hola a todos");
   N: Natural;
...
   N := ASU.Index (S, " ");      -- N tomará el valor 5
   ASU.Head (S, N-1);            -- S almacenará la cadena "Hola"
```

- Función `Tail`: Función que devuelve una parte de un `Unbounded_String`, comenzando por el final de dicho `Unbounded_String`.

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

  - `Source`: `Unbounded_String` del que se quiere extraer una parte 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, " ");               -- N tomará el valor 5
   R := ASU.Tail (S, ASU.Length(S)-N); -- R almacenará la cadena "a 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 una parte 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, el valor devuelto se completa con caracteres iguales a `Pad`.

  Ejemplo de uso:

```
   S: ASU.Unbounded_String := ASU.To_Unbounded_String("Hola a todos");
   N: Natural;
...
   N := ASU.Index (S, " ");          -- N tomará el valor 5
   ASU.Tail (S, ASU.Length(S)-N); -- S almacenará la cadena "a todos"
```

- Función `Index`[1]: Función que busca dentro de un `Unbounded_String` uno cualquiera de entre varios posibles caracteres, devolviendo la primera posición en la que aparece cualquiera de los caracteres buscados.

```
   function Index
     (Source  : Unbounded_String;
      Set     : Maps.Character_Set;
      Test    : Membership := Inside;
      Going   : Direction := Forward) return Natural;
```

  - `Source`: `Unbounded_String` en el que se busca.
  - `Set`: Conjunto de caracteres a buscar la primera aparición de uno cualquiera de ellos dentro de `Source`.
  - `Test`: Condición de pertenencia que debe cumplirse para dar la búsqueda por satisfactoria, por defecto que un caracter pertenezca al conjunto especificado en `Set`. Como este parámetro tiene un valor por defecto no es necesario pasarlo en la llamada.
  - `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.
  - Devuelve un `Natural` indicando la primera posición de `Source` en la que aparece por primera vez uno cualquiera de los caracteres especificados en `Set`. **Si no se encuentra ninguno, 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, Ada.Strings.Maps.To_Set(" ,.-");
                           -- Busca un espacio o coma o punto o guión
                           -- N tomará el valor 5
```

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.

[1] Sólo para extensión 1

8. 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 lee línea a línea un fichero de texto y va mostrando cada línea en la salida estándar:

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

procedure Show_File is
   package ACL renames Ada.Command_Line;
   package ASU renames Ada.Strings.Unbounded;

   Usage_Error: exception;

   File_Name: ASU.Unbounded_String;
   File: Ada.Text_IO.File_Type;
   Finish: Boolean;
   Line: ASU.Unbounded_String;

begin

   if ACL.Argument_Count /= 1 then
      raise Usage_Error;
   end if;

   File_Name := ASU.To_Unbounded_String(ACL.Argument(1));
   Ada.Text_IO.Open(File, Ada.Text_IO.In_File, ASU.To_String(File_Name));

   Finish := False;
   while not Finish loop
      begin
         Line := ASU.To_Unbounded_String(Ada.Text_IO.Get_Line(File));
         Ada.Text_IO.Put_Line(ASU.To_String(Line));
      exception
         when Ada.IO_Exceptions.End_Error =>
            Finish := True;
      end;
   end loop;

   Ada.Text_IO.Close(File);

exception
   when Usage_Error =>
      Ada.Text_IO.Put_Line("Use: ");
      Ada.Text_IO.Put_Line("       " & ACL.Command_Name & " <file>");
end Show_File;
```

9. Entrega

La entrega de esta práctica se hará de forma presencial en la siguiente fecha:

- **Alumnos de Programación de Sistemas de Telecomunicación**: Martes 6 de octubre a las 13h.
- **Alumnos de Informática II**: Miércoles 7 de octubre a las 13h.

A dicha entrega el alumno acudirá con el código de su práctica, y en el transcurso de la misma deberá ser capaz de realizar las modificaciones o extensiónes adicionales que se indiquen a su código o a otro similar que se le entregue.