Aclaraciones a la práctica de PRO2

Primavera 2019

17 de mayo de 2019

- 8-abr-2019: [Enunciado] Se cita la fuente de la figura y se corrigen pequeños errores
- 26-abr-2019: [Enunciado] Se explicita que el mínimo número de caracteres de un idioma es 2
- 30-abr-2019: [Jutge] En ocasiones, si la salida de un programa no coincide con la esperada (por ejemplo, si las strings de los nodos de un treecode se concatenan en orden incorrecto), puede obtenerse un veredicto IC (Invalid Character) en lugar de un WA (Wrong Answer); no obstante, si esto ocurre con el jp público, primero se han de descartar otras opciones más típicas de dicho veredicto
- 4-may-2019: [Ayuda] Cómo extraer uno a uno los caracteres de una string en el contexto de la práctica.

  Consideremos la string "seqüència". Parece claro que contiene 9 caracteres. Sin embargo, si le aplicamos la operación `length` (o `size`) el resultado es 11. La razón es que los caracteres especiales que contiene ('ü' y 'è') ocupan dos posiciones. A continuación, os proporcionamos un ejemplo de cómo extraer los caracteres de una string, formada por caracteres válidos según el enunciado de la práctica, para facilitar su codificación.

```
bool next_symbol(const string& s, int& i, string& out){
/* Pre: i=I<s.length() es la primera posicion de un caracter de s */
/* Post: out es el caracter que comienza en s[I]; si out es normal
   retorna true y i=I+1, si es especial retorna false y i=I+2 */

  if (s[i]>=0) {out = string(1, s[i]); ++i; return true;}
  else {out = string(s, i, 2); i+=2;; return false;}
}
```

```
string s;  cin >>s;
cout << s << " length: " << s.length() << endl;
int i=0;
while (i<s.length()){
  string aux;
  // los caracteres de s, uno a uno, se copian en aux,
  // se clasifican y se escriben
  bool b = next_symbol(s,i,aux);
  if (b) cout << aux << " normal" << endl;
  else cout << aux << " especial" << endl;
}
```

Si s es "seqüència" este código escribiría

```
seqüència length: 11
s normal
e normal
q normal
ü especial
è especial
n normal
c normal
i normal
a normal
```

La corrección de este método se basa en que todos los caracteres que son válidos para la práctica ocupan una o dos posiciones. Notad también que después de cada llamada a `next_symbol` se cumple que `i=s.length()` o `i` es la primera posición del caracter de `s` siguiente de `aux`.

- 17-may-2019: [Ayuda] Cómo evitar ineficiencias al concatenar strings.

  En los procesos para codificar y decodificar textos se han de ir concatenando pequeñas strings al final de la que contiene el texto resultante. Si x es dicho texto en un estado intermedio del proceso y s es la string que queremos concatenarle, podemos hacerlo de tres maneras, que en nuestro contexto presentan importantes diferencias

  - `x = x + s;` esto es ineficiente, pues equivale en esencia a producir una tercera string nueva a partir de las dos strings originales
  - `x += s;` esto es eficiente, solo añade los elementos de s a los ya existentes en x
  - `x.append(s)` versión funcional de la anterior, también eficiente