```
   1: ////////////////////////////////////////////////////////////////////////
   2: //
   3: // Fundamentos de Programación
   4: // ETS Informática y Telecomunicaciones
   5: // Universidad de Granada
   6: // Autor: Don Oreo
   7: //
   8: ////////////////////////////////////////////////////////////////////////
   9:
  10: // Secuencia de caracteres
  11:
  12: // IMPORTANTE:
  13: // La clase SecuenciaCaracteres es "atípica" en el sentido de que es una clase
  14: // con muchos métodos. El principio de responsabilidad única nos dice que
  15: // las clases deben tener una única responsabilidad y, por tanto, no suelen tener
  16: // un número elevado de métodos.
  17: // Sin embargo, a veces nos encontramos con este tipo de clases "genéricas"
  18: // En este caso, la responsabilidad es manejar una secuencia de caracteres
  19: // lo que conlleva la definición de numerosos métodos
  20: // La librería estándar STL contiene una clase (plantilla para ser más exactos)
  21: // similar a esta clase: es la plantilla denominada "vector"
  22: #include <iostream>
  23: #include <string>
  24: using namespace std;
  25:
  26: class SecuenciaCaracteres{
  27: private:
  28:    static const int TAMANIO = 2e6;  // 2e6 es un real (dos millones)
  29:                                     // -> casting automático a int
  30:
  31:                                     // Para poder dimensionar con un tamaño
  32:                                     // tan grande, hay que cambiar unos parámetros
  33:                                     // del compilador:
  34:                                     // Herramientas -> Opciones del Compilador ->
  35:                                     // Compilador -> Añadir las siguientes opciones
  36:                                     // -Wl,--stack,26000000
  37:    char v[TAMANIO];
  38:    int util;
  39:
  40:    void IntercambiaComponentesDirectamente(int pos_izda, int pos_dcha){
  41:       char intercambia;
  42:
  43:       intercambia = v[pos_izda];
  44:       v[pos_izda] = v[pos_dcha];
  45:       v[pos_dcha] = intercambia;
  46:    }
  47:
  48:    bool EsCorrectaPosicion(int indice){
  49:       return 0 <= indice && indice < util;
  50:    }
  51: public:
  52:    /*//Si lo ponemos publico, debemos comprobrar si los datos proporcionados son correctos(en private se
puede omitir)
  53:     void IntercambiaComponentesDirectamente(int pos_izda, int pos_dcha){
  54:       char intercambia;
  55:       if(EsCorrectaPosicion(pos_izda)
  56:          &&
  57:          EsCorrectaPosicion(pos_dcha)){
  58:
  59:          intercambia = v[pos_izda];
  60:          v[pos_izda] = v[pos_dcha];
  61:          v[pos_dcha] = intercambia;
  62:       }
  63:    }
  64:    */
  65:    SecuenciaCaracteres()
  66:       :util(0) {
  67:    }
  68:
  69:    int Utilizados(){
  70:       return util;
  71:    }
  72:
  73:    int Capacidad(){
  74:       return TAMANIO;
  75:    }
  76:
  77:    void EliminaTodos(){
  78:       util = 0;
  79:    }
  80:
  81:    void Aniade(char nuevo){
  82:       if (util < TAMANIO){
  83:          v[util] = nuevo;
  84:          util++;
  85:       }
```

```
   }

   void Modifica(int posicion, char nuevo){
      if (EsCorrectaPosicion(posicion))
         v[posicion] = nuevo;
   }

   char Elemento(int indice){
      return v[indice];
   }

   string ToString(){
      // Si el número de caracteres en memoria es muy grande,
      // es mucho más eficiente reservar memoria previamente
      // y usar push_back

      string cadena;

      cadena.reserve(util);

      for (int i=0; i < util; i++)
         cadena.push_back(v[i]);
         //cadena = cadena + v[i]  <- Evitarlo. Muy ineficiente para tamaños grandes;

      return cadena;
   }

   int PrimeraOcurrenciaEntre (int pos_izda, int pos_dcha, char buscado){
      int i = pos_izda;
      bool encontrado = false;

      while (i <= pos_dcha  &&  !encontrado)
         if (v[i] == buscado)
            encontrado = true;
         else
            i++;

      if (encontrado)
         return i;
      else
         return -1;
   }

   int PrimeraOcurrencia (char buscado){
      return PrimeraOcurrenciaEntre (0, util - 1, buscado);
   }


   ////////////////////////////////////////////////////////////
   // Búsquedas

   // Precond: 0 <= izda <= dcha < util
   int PosMinimoEntre(int izda, int dcha){
      int pos_minimo = -1;
      char minimo;

      minimo = v[izda];
      pos_minimo = izda;

      for (int i = izda+1 ; i <= dcha ; i++)
         if (v[i] < minimo){
            minimo = v[i];
            pos_minimo = i;
         }

      return pos_minimo;
   }

   int PosMinimo(){
      return PosMinimoEntre(0, util - 1);
   }

   int BusquedaBinaria (char buscado){
      int izda, dcha, centro;
      bool encontrado = false;

      izda = 0;
      dcha = util - 1;
      centro = (izda + dcha) / 2;

      while (izda <= dcha  &&  !encontrado){
         if (v[centro] == buscado)
            encontrado = true;
         else if (buscado < v[centro])
            dcha = centro - 1;
         else
```

```
172:             izda = centro + 1;
173:
174:          centro = (izda + dcha) / 2;
175:       }
176:
177:       if (encontrado)
178:          return centro;
179:       else
180:          return -1;
181:    }
182:
183:
184:    ////////////////////////////////////////////////////////
185:    // Recorridos que modifican las componentes
186:
187:    // Inserta un valor en la posición especificada
188:    void Inserta(int pos_insercion, char nuevo){
189:       if (util < TAMANIO  &&  pos_insercion >= 0
190:          &&  pos_insercion <= util){
191:
192:          for (int i = util ; i > pos_insercion ; i--)
193:             v[i] = v[i-1];
194:
195:          v[pos_insercion] = nuevo;
196:          util++;
197:       }
198:    }
199:
200:    /*
201:    Tipos de borrados:
202:       - Lógico
203:          Usar un valor de componente especial y marcar la componente con dicho valor
204:          Un vector de edades -> valor -1
205:          Un vector de caracteres alfabéticos -> '@'
206:          Ventajas: Muy rápido
207:
208:          Inconvenientes: Cualquier procesado posterior del vector
209:          debe tratar las componentes marcadas de una forma especial
210:
211:       - Físico
212:          Implica desplazar 1 posición a la izquierda, todas las componentes que hay a la derecha de
213:          la que queremos borrar.
214:
215:          Tiene justo las ventajas e incovenientes contrarias que el método anterior.
216:
217:        En esta versión, implementamos el borrado físico.
218:    */
219:
220:    // Elimina una componente, dada por su posición
221:    void Elimina (int posicion){
222:       /*
223:       Algoritmo:
224:
225:          Recorremos de izquierda a derecha toda las componentes
226:          que hay a la derecha de la posición a eliminar
227:             Le asignamos a cada componente la que hay a su derecha
228:       */
229:       if (posicion >= 0 && posicion < util){
230:          int tope = util-1;
231:
232:          for (int i = posicion ; i < tope ; i++)
233:             v[i] = v[i+1];
234:
235:          util--;
236:       }
237:
238:       // Nota:
239:
240:       // En vez de usar la asignación
241:       //    v[i] = v[i+1];
242:       // también podríamos haber puesto lo siguiente:
243:       //    Modifica(i, Elemento(i+1));
244:       // Hemos preferido acceder directamente a las componentes con la notación en corchete
245:       // para aumentar la eficiencia del método Elimina, ya que si el vector es muy grande
246:       // tendrá que realizar muchos desplazamientos y, por tanto, muchos accesos al método
247:       // Elemento. En general, desde dentro de la clase, los métodos de la clase Secuencia
248:       // accederán directamente a las componentes con la notación corchete
249:
250:       // Además, cuando entramos en la función Elimina, comprobamos con el condicional
251:       // que los accesos a los índices son correctos.
252:       // Si usamos el método Modifica, volveríamos a comprobar lo mismo.
253:
254:       // Nota:
255:
256:       // ¿Y si en vez de asignar v[i] = v[i+1];
257:       // llamamos a IntercambiaComponentesDirectamente(i, i+1) ?
```

```
258:       // La componente se eliminaría pero realizando el doble de asignaciones
259:       // Obviamente, no es necesario intercambiar las componentes.
260:       // Únicamente debemos ir asignando v[i] = v[i+1] de izquierda a derecha.
261:    }
262:
263:
264:    ////////////////////////////////////////////////////////
265:    // Algoritmos de ordenación
266:
267:    void Ordena_por_Seleccion(){
268:       int pos_min;
269:
270:       for (int izda = 0 ; izda < util ; izda++){
271:          pos_min = PosMinimoEntre(izda, util - 1);
272:          IntercambiaComponentesDirectamente(izda, pos_min);
273:       }
274:    }
275:
276:    void Ordena_por_Insercion(){
277:       int izda, i;
278:       char a_desplazar;
279:
280:       for (izda=1; izda < util; izda++){
281:          a_desplazar = v[izda];
282:
283:          for (i=izda; i > 0 && a_desplazar < v[i-1]; i--)
284:             v[i] = v[i-1];
285:
286:          v[i] = a_desplazar;
287:       }
288:    }
289:
290:    void InsertaOrdenadamente(char nuevo){
291:         int i;
292:
293:         if (util > TAMANIO){
294:            for (i=util; i>0 && nuevo < v[i-1]; i--)
295:                 v[i] = v[i-1];
296:
297:            v[i] = nuevo;
298:            util++;
299:         }
300:    }
301:
302:
303:    void Ordena_por_Burbuja(){
304:       int izda, i;
305:
306:       for (izda = 0; izda < util; izda++)
307:         for (i = util-1 ; i > izda ; i--)
308:           if (v[i] < v[i-1])
309:              IntercambiaComponentesDirectamente(i, i-1);
310:    }
311:
312:    void Ordena_por_BurbujaMejorado(){
313:       int izda, i;
314:       bool cambio;
315:
316:       cambio= true;
317:
318:       for (izda=0; izda < util && cambio; izda++){
319:         cambio=false;
320:
321:         for (i=util-1 ; i>izda ; i--)
322:           if (v[i] < v[i-1]){
323:              IntercambiaComponentesDirectamente(i, i-1);
324:              cambio=true;
325:           }
326:       }
327:    }
328:
329:    void AniadeVarios(SecuenciaCaracteres nuevos){
330:       int totales_a_aniadir = nuevos.Utilizados();
331:
332:       for (int i = 0; i < totales_a_aniadir; i++)
333:          Aniade(nuevos.Elemento(i)); // Es importante entender
334:    }
335:
336:
337:    SecuenciaCaracteres ToUpper(){
338:       SecuenciaCaracteres en_mayuscula;
339:
340:       for(int i = 0; i < util; i++)
341:          en_mayuscula.Aniade(toupper(v[i]));
342:
343:       return en_mayuscula;
```

```
344:    }
345:    bool EsPalindromo(){
346:       int izda = 0;
347:       int dcha = util - 1;
348:       bool es_palindromo = true;
349:       while(izda < dcha && es_palindromo){
350:          if(v[izda] != v[dcha])
351:             es_palindromo = false;
352:          izda++;
353:          dcha--;
354:       }
355:    return es_palindromo;
356:    }
357:
358:    void Invierte(){
359:       int izda = 0;
360:       int dcha = util - 1;
361:       bool es_palindromo;
362:
363:       es_palindromo = EsPalindromo();
364:
365:       while(izda < dcha && !es_palindromo){
366:          IntercambiaComponentesDirectamente(izda,dcha);
367:          izda++;
368:          dcha--;
369:       }
370:       cout << "El vector invertido es: ";
371:       for (int i = 0; i < util; i++)
372:          cout  << v[i];
373:    }
374:
375: };
376:
377:
378: int main(){
379:    SecuenciaCaracteres cadena;
380:    char car;
381:    bool es_palindromo;
382:
383:    car = cin.get();
384:    while(car != '\n'){      //Pide caracteres hasta introducir 'enter'
385:       cadena.Aniade(car);
386:       car = cin.get();
387:    }
388:
389:    es_palindromo = cadena.EsPalindromo();
390:
391:    if(es_palindromo)
392:       cout << "Es palindromo" << endl;
393:    else
394:       cout << "No es palindromo" << endl;
395:
396:    cadena.Invierte();
397: }
398:
```

```
1: ////////////////////////////////////////////////////////////////////////
2: //
3: // Fundamentos de Programación
4: // ETS Informática y Telecomunicaciones
5: // Universidad de Granada
6: // Autor: Don Oreo
7: //
8: ////////////////////////////////////////////////////////////////////////
9:
10: // Secuencia de caracteres
11:
12: // IMPORTANTE:
13: // La clase SecuenciaCaracteres es "atípica" en el sentido de que es una clase
14: // con muchos métodos. El principio de responsabilidad única nos dice que
15: // las clases deben tener una única responsabilidad y, por tanto, no suelen tener
16: // un número elevado de métodos.
17: // Sin embargo, a veces nos encontramos con este tipo de clases "genéricas"
18: // En este caso, la responsabilidad es manejar una secuencia de caracteres
19: // lo que conlleva la definición de numerosos métodos
20: // La librería estándar STL contiene una clase (plantilla para ser más exactos)
21: // similar a esta clase: es la plantilla denominada "vector"
22:
23: #include <iostream>
24: #include <string>
25: using namespace std;
26:
27: class SecuenciaCaracteres{
28: private:
29:    static const int TAMANIO = 2e6;  // 2e6 es un real (dos millones)
30:                                     // -> casting automático a int
31:
32:                                     // Para poder dimensionar con un tamaño
33:                                     // tan grande, hay que cambiar unos parámetros
34:                                     // del compilador:
35:                                     // Herramientas -> Opciones del Compilador ->
36:                                     // Compilador -> Añadir las siguientes opciones
37:                                     // -Wl,--stack,26000000
38:    char v[TAMANIO];
39:    int util;
40:
41:    void IntercambiaComponentesDirectamente(int pos_izda, int pos_dcha){
42:       char intercambia;
43:
44:       intercambia = v[pos_izda];
45:       v[pos_izda] = v[pos_dcha];
46:       v[pos_dcha] = intercambia;
47:    }
48:
49:    bool EsCorrectaPosicion(int indice){
50:       return 0 <= indice && indice < util;
51:    }
52: public:
53:    SecuenciaCaracteres()
54:       :util(0) {
55:    }
56:
57:    int Utilizados(){
58:       return util;
59:    }
60:
61:    int Capacidad(){
62:       return TAMANIO;
63:    }
64:
65:    void EliminaTodos(){
66:       util = 0;
67:    }
68:
69:    void Aniade(char nuevo){
70:       if (util < TAMANIO){
71:          v[util] = nuevo;
72:          util++;
73:       }
74:    }
75:
76:    void Modifica(int posicion, char nuevo){
77:       if (EsCorrectaPosicion(posicion))
78:          v[posicion] = nuevo;
79:    }
80:
81:    char Elemento(int indice){
82:       return v[indice];
83:    }
84:
85:    string ToString(){
86:       // Si el número de caracteres en memoria es muy grande,
```

```
87:       // es mucho más eficiente reservar memoria previamente
88:       // y usar push_back
89:
90:       string cadena;
91:
92:       cadena.reserve(util);
93:
94:       for (int i=0; i < util; i++)
95:          cadena.push_back(v[i]);
96:          //cadena = cadena + v[i]  <- Evitarlo. Muy ineficiente para tamaños grandes;
97:
98:       return cadena;
99:    }
100:
101:   int PrimeraOcurrenciaEntre (int pos_izda, int pos_dcha, char buscado){
102:      int i = pos_izda;
103:      bool encontrado = false;
104:
105:      while (i <= pos_dcha  &&  !encontrado)
106:         if (v[i] == buscado)
107:            encontrado = true;
108:         else
109:            i++;
110:
111:      if (encontrado)
112:         return i;
113:      else
114:         return -1;
115:   }
116:
117:   int PrimeraOcurrencia (char buscado){
118:      return PrimeraOcurrenciaEntre (0, util - 1, buscado);
119:   }
120:
121:
122:   ////////////////////////////////////////////////////////////
123:   // Búsquedas
124:
125:   // Precond: 0 <= izda <= dcha < util
126:   int PosMinimoEntre(int izda, int dcha){
127:      int pos_minimo = -1;
128:      char minimo;
129:
130:      minimo = v[izda];
131:      pos_minimo = izda;
132:
133:      for (int i = izda+1 ; i <= dcha ; i++)
134:         if (v[i] < minimo){
135:            minimo = v[i];
136:            pos_minimo = i;
137:         }
138:
139:      return pos_minimo;
140:   }
141:
142:   int PosMinimo(){
143:      return PosMinimoEntre(0, util - 1);
144:   }
145:
146:   int BusquedaBinaria (char buscado){
147:      int izda, dcha, centro;
148:      bool encontrado = false;
149:
150:      izda = 0;
151:      dcha = util - 1;
152:      centro = (izda + dcha) / 2;
153:
154:      while (izda <= dcha  &&  !encontrado){
155:         if (v[centro] == buscado)
156:            encontrado = true;
157:         else if (buscado < v[centro])
158:            dcha = centro - 1;
159:         else
160:            izda = centro + 1;
161:
162:         centro = (izda + dcha) / 2;
163:      }
164:
165:      if (encontrado)
166:         return centro;
167:      else
168:         return -1;
169:   }
170:
171:
172:   ////////////////////////////////////////////////////////////
```

```
173:    // Recorridos que modifican las componentes
174:
175:    // Inserta un valor en la posición especificada
176:    void Inserta(int pos_insercion, char nuevo){
177:       if (util < TAMANIO  &&  pos_insercion >= 0
178:          &&  pos_insercion <= util){
179:
180:          for (int i = util ; i > pos_insercion ; i--)
181:             v[i] = v[i-1];
182:
183:          v[pos_insercion] = nuevo;
184:          util++;
185:       }
186:    }
187:
188:    /*
189:    Tipos de borrados:
190:       - Lógico
191:          Usar un valor de componente especial y marcar la componente con dicho valor
192:          Un vector de edades -> valor -1
193:          Un vector de caracteres alfabéticos -> '@'
194:          Ventajas: Muy rápido
195:
196:          Inconvenientes: Cualquier procesado posterior del vector
197:          debe tratar las componentes marcadas de una forma especial
198:
199:       - Físico
200:          Implica desplazar 1 posición a la izquierda, todas las componentes que hay a la derecha de
201:          la que queremos borrar.
202:
203:          Tiene justo las ventajas e incovenientes contrarias que el método anterior.
204:
205:        En esta versión, implementamos el borrado físico.
206:    */
207:
208:    // Elimina una componente, dada por su posición
209:    void Elimina (int posicion){
210:       /*
211:       Algoritmo:
212:
213:          Recorremos de izquierda a derecha toda las componentes
214:          que hay a la derecha de la posición a eliminar
215:             Le asignamos a cada componente la que hay a su derecha
216:       */
217:       if (posicion >= 0 && posicion < util){
218:          int tope = util-1;
219:
220:          for (int i = posicion ; i < tope ; i++)
221:             v[i] = v[i+1];
222:
223:          util--;
224:       }
225:
226:       // Nota:
227:
228:       // En vez de usar la asignación
229:       //    v[i] = v[i+1];
230:       // también podríamos haber puesto lo siguiente:
231:       //    Modifica(i, Elemento(i+1));
232:       // Hemos preferido acceder directamente a las componentes con la notación en corchete
233:       // para aumentar la eficiencia del método Elimina, ya que si el vector es muy grande
234:       // tendrá que realizar muchos desplazamientos y, por tanto, muchos accesos al método
235:       // Elemento. En general, desde dentro de la clase, los métodos de la clase Secuencia
236:       // accederán directamente a las componentes con la notación corchete
237:
238:       // Además, cuando entramos en la función Elimina, comprobamos con el condicional
239:       // que los accesos a los índices son correctos.
240:       // Si usamos el método Modifica, volveríamos a comprobar lo mismo.
241:
242:       // Nota:
243:
244:       // ¿Y si en vez de asignar v[i] = v[i+1];
245:       // llamamos a IntercambiaComponentesDirectamente(i, i+1) ?
246:       // La componente se eliminaría pero realizando el doble de asignaciones
247:       // Obviamente, no es necesario intercambiar las componentes.
248:       // Únicamente debemos ir asignando v[i] = v[i+1] de izquierda a derecha.
249:    }
250:
251:    void EliminaOcurrencias(char a_borrar){
252:       for(int i = util; i >= 0; i--)
253:          if(v[i] == a_borrar)
254:             Elimina(i);
255:
256:       cout << "El vector sin a_borrar: ";
257:       for(int i = 0; i < util; i++)
258:          cout << v[i];
```

```
259:    }
260:
261:    ////////////////////////////////////////////////////////////
262:    // Algoritmos de ordenación
263:
264:    void Ordena_por_Seleccion(){
265:       int pos_min;
266:
267:       for (int izda = 0 ; izda < util ; izda++){
268:          pos_min = PosMinimoEntre(izda, util - 1);
269:          IntercambiaComponentesDirectamente(izda, pos_min);
270:       }
271:    }
272:
273:    void Ordena_por_Insercion(){
274:       int izda, i;
275:       char a_desplazar;
276:
277:       for (izda=1; izda < util; izda++){
278:          a_desplazar = v[izda];
279:
280:          for (i=izda; i > 0 && a_desplazar < v[i-1]; i--)
281:             v[i] = v[i-1];
282:
283:          v[i] = a_desplazar;
284:       }
285:    }
286:
287:    void InsertaOrdenadamente(char nuevo){
288:         int i;
289:
290:         if (util > TAMANIO){
291:            for (i=util; i>0 && nuevo < v[i-1]; i--)
292:                 v[i] = v[i-1];
293:
294:            v[i] = nuevo;
295:            util++;
296:         }
297:    }
298:
299:
300:    void Ordena_por_Burbuja(){
301:       int izda, i;
302:
303:       for (izda = 0; izda < util; izda++)
304:         for (i = util-1 ; i > izda ; i--)
305:           if (v[i] < v[i-1])
306:              IntercambiaComponentesDirectamente(i, i-1);
307:    }
308:
309:    void Ordena_por_BurbujaMejorado(){
310:       int izda, i;
311:       bool cambio;
312:
313:       cambio= true;
314:
315:       for (izda=0; izda < util && cambio; izda++){
316:         cambio=false;
317:
318:         for (i=util-1 ; i>izda ; i--)
319:           if (v[i] < v[i-1]){
320:              IntercambiaComponentesDirectamente(i, i-1);
321:              cambio=true;
322:           }
323:       }
324:    }
325:
326:    void AniadeVarios(SecuenciaCaracteres nuevos){
327:       int totales_a_aniadir = nuevos.Utilizados();
328:
329:       for (int i = 0; i < totales_a_aniadir; i++)
330:          Aniade(nuevos.Elemento(i)); // Es importante entender
331:    }
332:
333:
334:    SecuenciaCaracteres ToUpper(){
335:       SecuenciaCaracteres en_mayuscula;
336:
337:       for(int i = 0; i < util; i++)
338:          en_mayuscula.Aniade(toupper(v[i]));
339:
340:       return en_mayuscula;
341:    }
342: };
343:
344: int main(){
```

```
345:     SecuenciaCaracteres cadena;
346:     char car,a_borrar;
347:
348:     car = cin.get();
349:     while(car != '#'){     //Pide caracteres hasta introducir '#'
350:        cadena.Aniade(car);
351:        car = cin.get();
352:     }
353:
354:     a_borrar = cin.get();
355:
356:     cadena.EliminaOcurrencias(a_borrar);
357:
358:     return 0;
359: }
360:
```

```
1: ////////////////////////////////////////////////////////////////////////
2: //
3: // Fundamentos de Programación
4: // ETS Informática y Telecomunicaciones
5: // Universidad de Granada
6: // Autor: Don Oreo
7: //
8: ////////////////////////////////////////////////////////////////////////
9:
10: // Secuencia de caracteres
11:
12: // IMPORTANTE:
13: // La clase SecuenciaCaracteres es "atípica" en el sentido de que es una clase
14: // con muchos métodos. El principio de responsabilidad única nos dice que
15: // las clases deben tener una única responsabilidad y, por tanto, no suelen tener
16: // un número elevado de métodos.
17: // Sin embargo, a veces nos encontramos con este tipo de clases "genéricas"
18: // En este caso, la responsabilidad es manejar una secuencia de caracteres
19: // lo que conlleva la definición de numerosos métodos
20: // La librería estándar STL contiene una clase (plantilla para ser más exactos)
21: // similar a esta clase: es la plantilla denominada "vector"
22:
23: #include <iostream>
24: #include <string>
25: using namespace std;
26:
27: class SecuenciaCaracteres{
28: private:
29:    static const int TAMANIO = 3e6;  // 2e6 es un real (dos millones)
30:                                     // -> casting automático a int
31:
32:                                     // Para poder dimensionar con un tamaño
33:                                     // tan grande, hay que cambiar unos parámetros
34:                                     // del compilador:
35:                                     // Herramientas -> Opciones del Compilador ->
36:                                     // Compilador -> Añadir las siguientes opciones
37:                                     // -Wl,--stack,26000000
38:    char v[TAMANIO];
39:    int util;
40:
41:    void IntercambiaComponentesDirectamente(int pos_izda, int pos_dcha){
42:       char intercambia;
43:
44:       intercambia = v[pos_izda];
45:       v[pos_izda] = v[pos_dcha];
46:       v[pos_dcha] = intercambia;
47:    }
48:
49:    bool EsCorrectaPosicion(int indice){
50:       return 0 <= indice && indice < util;
51:    }
52: public:
53:    SecuenciaCaracteres()
54:       :util(0) {
55:    }
56:
57:    int Utilizados(){
58:       return util;
59:    }
60:
61:    int Capacidad(){
62:       return TAMANIO;
63:    }
64:
65:    void EliminaTodos(){
66:       util = 0;
67:    }
68:
69:    void Aniade(char nuevo){
70:       if (util < TAMANIO){
71:          v[util] = nuevo;
72:          util++;
73:       }
74:    }
75:
76:    void Modifica(int posicion, char nuevo){
77:       if (EsCorrectaPosicion(posicion))
78:          v[posicion] = nuevo;
79:    }
80:
81:    char Elemento(int indice){
82:       return v[indice];
83:    }
84:
85:    string ToString(){
86:       // Si el número de caracteres en memoria es muy grande,
```

```
87:       // es mucho más eficiente reservar memoria previamente
88:       // y usar push_back
89:
90:       string cadena;
91:
92:       cadena.reserve(util);
93:
94:       for (int i=0; i < util; i++)
95:          cadena.push_back(v[i]);
96:          //cadena = cadena + v[i]  <- Evitarlo. Muy ineficiente para tamaños grandes;
97:
98:       return cadena;
99:    }
100:
101:   int PrimeraOcurrenciaEntre (int pos_izda, int pos_dcha, char buscado){
102:      int i = pos_izda;
103:      bool encontrado = false;
104:
105:      while (i <= pos_dcha  &&  !encontrado)
106:         if (v[i] == buscado)
107:            encontrado = true;
108:         else
109:            i++;
110:
111:      if (encontrado)
112:         return i;
113:      else
114:         return -1;
115:   }
116:
117:   int PrimeraOcurrencia (char buscado){
118:      return PrimeraOcurrenciaEntre (0, util - 1, buscado);
119:   }
120:
121:
122:   ////////////////////////////////////////////////////////////
123:   // Búsquedas
124:
125:   // Precond: 0 <= izda <= dcha < util
126:   int PosMinimoEntre(int izda, int dcha){
127:      int pos_minimo = -1;
128:      char minimo;
129:
130:      minimo = v[izda];
131:      pos_minimo = izda;
132:
133:      for (int i = izda+1 ; i <= dcha ; i++)
134:         if (v[i] < minimo){
135:            minimo = v[i];
136:            pos_minimo = i;
137:         }
138:
139:      return pos_minimo;
140:   }
141:
142:   int PosMinimo(){
143:      return PosMinimoEntre(0, util - 1);
144:   }
145:
146:   int BusquedaBinaria (char buscado){
147:      int izda, dcha, centro;
148:      bool encontrado = false;
149:
150:      izda = 0;
151:      dcha = util - 1;
152:      centro = (izda + dcha) / 2;
153:
154:      while (izda <= dcha  &&  !encontrado){
155:         if (v[centro] == buscado)
156:            encontrado = true;
157:         else if (buscado < v[centro])
158:            dcha = centro - 1;
159:         else
160:            izda = centro + 1;
161:
162:         centro = (izda + dcha) / 2;
163:      }
164:
165:      if (encontrado)
166:         return centro;
167:      else
168:         return -1;
169:   }
170:
171:
172:   ////////////////////////////////////////////////////////////
```

```
173:    // Recorridos que modifican las componentes
174:
175:    // Inserta un valor en la posición especificada
176:    void Inserta(int pos_insercion, char nuevo){
177:       if (util < TAMANIO  &&  pos_insercion >= 0
178:          &&  pos_insercion <= util){
179:
180:          for (int i = util ; i > pos_insercion ; i--)
181:             v[i] = v[i-1];
182:
183:          v[pos_insercion] = nuevo;
184:          util++;
185:       }
186:    }
187:
188:    /*
189:    Tipos de borrados:
190:       - Lógico
191:          Usar un valor de componente especial y marcar la componente con dicho valor
192:          Un vector de edades -> valor -1
193:          Un vector de caracteres alfabéticos -> '@'
194:          Ventajas: Muy rápido
195:
196:          Inconvenientes: Cualquier procesado posterior del vector
197:          debe tratar las componentes marcadas de una forma especial
198:
199:       - Físico
200:          Implica desplazar 1 posición a la izquierda, todas las componentes que hay a la derecha de
201:          la que queremos borrar.
202:
203:          Tiene justo las ventajas e incovenientes contrarias que el método anterior.
204:
205:        En esta versión, implementamos el borrado físico.
206:    */
207:
208:    // Elimina una componente, dada por su posición
209:    void Elimina (int posicion){
210:       /*
211:       Algoritmo:
212:
213:          Recorremos de izquierda a derecha toda las componentes
214:          que hay a la derecha de la posición a eliminar
215:             Le asignamos a cada componente la que hay a su derecha
216:       */
217:       if (posicion >= 0 && posicion < util){
218:          int tope = util-1;
219:
220:          for (int i = posicion ; i < tope ; i++)
221:             v[i] = v[i+1];
222:
223:          util--;
224:       }
225:    }
226:
227:    void EliminaOcurrencias(char a_borrar){
228:       int escr, lect;
229:       lect = escr = 0;
230:
231:       while (v[lect] != a_borrar){
232:          lect++;
233:          escr++;
234:       }
235:
236:       while (lect < util){
237:          if (v[lect] != a_borrar){
238:             v[escr] = v[lect];
239:             escr++;
240:          }
241:
242:          lect++;
243:       }
244:       util = escr;
245:
246:       cout << "El vector sin a_borrar: ";
247:       for(int i = 0; i < util; i++)
248:          cout << v[i];
249:    }
250:
251:    ////////////////////////////////////////////////////////
252:    // Algoritmos de ordenación
253:
254:    void Ordena_por_Seleccion(){
255:       int pos_min;
256:
257:       for (int izda = 0 ; izda < util ; izda++){
258:          pos_min = PosMinimoEntre(izda, util - 1);
```

```
259:          IntercambiaComponentesDirectamente(izda, pos_min);
260:       }
261:    }
262:
263:    void Ordena_por_Insercion(){
264:       int izda, i;
265:       char a_desplazar;
266:
267:       for (izda=1; izda < util; izda++){
268:          a_desplazar = v[izda];
269:
270:          for (i=izda; i > 0 && a_desplazar < v[i-1]; i--)
271:             v[i] = v[i-1];
272:
273:          v[i] = a_desplazar;
274:       }
275:    }
276:
277:    void InsertaOrdenadamente(char nuevo){
278:         int i;
279:
280:         if (util > TAMANIO){
281:            for (i=util; i>0 && nuevo < v[i-1]; i--)
282:                 v[i] = v[i-1];
283:
284:            v[i] = nuevo;
285:            util++;
286:         }
287:    }
288:
289:
290:    void Ordena_por_Burbuja(){
291:       int izda, i;
292:
293:       for (izda = 0; izda < util; izda++)
294:         for (i = util-1 ; i > izda ; i--)
295:           if (v[i] < v[i-1])
296:              IntercambiaComponentesDirectamente(i, i-1);
297:    }
298:
299:    void Ordena_por_BurbujaMejorado(){
300:       int izda, i;
301:       bool cambio;
302:
303:       cambio= true;
304:
305:       for (izda=0; izda < util && cambio; izda++){
306:         cambio=false;
307:
308:         for (i=util-1 ; i>izda ; i--)
309:           if (v[i] < v[i-1]){
310:              IntercambiaComponentesDirectamente(i, i-1);
311:              cambio=true;
312:           }
313:       }
314:    }
315:
316:    void AniadeVarios(SecuenciaCaracteres nuevos){
317:       int totales_a_aniadir = nuevos.Utilizados();
318:
319:       for (int i = 0; i < totales_a_aniadir; i++)
320:          Aniade(nuevos.Elemento(i)); // Es importante entender
321:    }
322:
323:
324:    SecuenciaCaracteres ToUpper(){
325:       SecuenciaCaracteres en_mayuscula;
326:
327:       for(int i = 0; i < util; i++)
328:          en_mayuscula.Aniade(toupper(v[i]));
329:
330:       return en_mayuscula;
331:    }
332: };
333:
334: int main(){
335:    SecuenciaCaracteres cadena;
336:    char car,a_borrar;
337:
338:    car = cin.get();
339:    while(car != '#'){     //Pide caracteres hasta introducir '#'
340:       cadena.Aniade(car);
341:       car = cin.get();
342:    }
343:
344:    a_borrar = cin.get();
```

```
345:
346:    cadena.EliminaOcurrencias(a_borrar);
347:
348:    return 0;
349: }
350:
```

```cpp
////////////////////////////////////////////////////////////////////////
//
// Fundamentos de Programación
// ETS Informática y Telecomunicaciones
// Universidad de Granada
// Autor: Don Oreo
//
////////////////////////////////////////////////////////////////////////

// Secuencia de caracteres

// IMPORTANTE:
// La clase SecuenciaCaracteres es "atípica" en el sentido de que es una clase
// con muchos métodos. El principio de responsabilidad única nos dice que
// las clases deben tener una única responsabilidad y, por tanto, no suelen tener
// un número elevado de métodos.
// Sin embargo, a veces nos encontramos con este tipo de clases "genéricas"
// En este caso, la responsabilidad es manejar una secuencia de caracteres
// lo que conlleva la definición de numerosos métodos
// La librería estándar STL contiene una clase (plantilla para ser más exactos)
// similar a esta clase: es la plantilla denominada "vector"

#include <iostream>
#include <string>
using namespace std;

class SecuenciaCaracteres{
private:
   static const int TAMANIO = 2e6;  // 2e6 es un real (dos millones)
                                    // -> casting automático a int

                                    // Para poder dimensionar con un tamaño
                                    // tan grande, hay que cambiar unos parámetros
                                    // del compilador:
                                    // Herramientas -> Opciones del Compilador ->
                                    // Compilador -> Añadir las siguientes opciones
                                    // -Wl,--stack,26000000
   char v[TAMANIO];
   int util;

   void IntercambiaComponentesDirectamente(int pos_izda, int pos_dcha){
      char intercambia;

      intercambia = v[pos_izda];
      v[pos_izda] = v[pos_dcha];
      v[pos_dcha] = intercambia;
   }

   bool EsCorrectaPosicion(int indice){
      return 0 <= indice && indice < util;
   }
public:
   SecuenciaCaracteres()
      :util(0) {
   }

   int Utilizados(){
      return util;
   }

   int Capacidad(){
      return TAMANIO;
   }

   void EliminaTodos(){
      util = 0;
   }

   void Aniade(char nuevo){
      if (util < TAMANIO){
         v[util] = nuevo;
         util++;
      }
   }

   void Modifica(int posicion, char nuevo){
      if (EsCorrectaPosicion(posicion))
         v[posicion] = nuevo;
   }

   char Elemento(int indice){
      return v[indice];
   }

   string ToString(){
      // Si el número de caracteres en memoria es muy grande,
```

```
87:       // es mucho más eficiente reservar memoria previamente
88:       // y usar push_back
89:
90:       string cadena;
91:
92:       cadena.reserve(util);
93:
94:       for (int i=0; i < util; i++)
95:          cadena.push_back(v[i]);
96:          //cadena = cadena + v[i]  <- Evitarlo. Muy ineficiente para tamaños grandes;
97:
98:       return cadena;
99:    }
100:
101:   int PrimeraOcurrenciaEntre (int pos_izda, int pos_dcha, char buscado){
102:      int i = pos_izda;
103:      bool encontrado = false;
104:
105:      while (i <= pos_dcha  &&  !encontrado)
106:         if (v[i] == buscado)
107:            encontrado = true;
108:         else
109:            i++;
110:
111:      if (encontrado)
112:         return i;
113:      else
114:         return -1;
115:   }
116:
117:   int PrimeraOcurrencia (char buscado){
118:      return PrimeraOcurrenciaEntre (0, util - 1, buscado);
119:   }
120:
121:
122:   ////////////////////////////////////////////////////////////
123:   // Búsquedas
124:
125:   // Precond: 0 <= izda <= dcha < util
126:   int PosMinimoEntre(int izda, int dcha){
127:      int pos_minimo = -1;
128:      char minimo;
129:
130:      minimo = v[izda];
131:      pos_minimo = izda;
132:
133:      for (int i = izda+1 ; i <= dcha ; i++)
134:         if (v[i] < minimo){
135:            minimo = v[i];
136:            pos_minimo = i;
137:         }
138:
139:      return pos_minimo;
140:   }
141:
142:   int PosMinimo(){
143:      return PosMinimoEntre(0, util - 1);
144:   }
145:
146:   int BusquedaBinaria (char buscado){
147:      int izda, dcha, centro;
148:      bool encontrado = false;
149:
150:      izda = 0;
151:      dcha = util - 1;
152:      centro = (izda + dcha) / 2;
153:
154:      while (izda <= dcha  &&  !encontrado){
155:         if (v[centro] == buscado)
156:            encontrado = true;
157:         else if (buscado < v[centro])
158:            dcha = centro - 1;
159:         else
160:            izda = centro + 1;
161:
162:         centro = (izda + dcha) / 2;
163:      }
164:
165:      if (encontrado)
166:         return centro;
167:      else
168:         return -1;
169:   }
170:
171:
172:   ////////////////////////////////////////////////////////////
```

```
173:    // Recorridos que modifican las componentes
174:
175:    // Inserta un valor en la posición especificada
176:    void Inserta(int pos_insercion, char nuevo){
177:       if (util < TAMANIO  &&  pos_insercion >= 0
178:          &&  pos_insercion <= util){
179:
180:          for (int i = util ; i > pos_insercion ; i--)
181:             v[i] = v[i-1];
182:
183:          v[pos_insercion] = nuevo;
184:          util++;
185:       }
186:    }
187:
188:    /*
189:    Tipos de borrados:
190:       - Lógico
191:          Usar un valor de componente especial y marcar la componente con dicho valor
192:          Un vector de edades -> valor -1
193:          Un vector de caracteres alfabéticos -> '@'
194:          Ventajas: Muy rápido
195:
196:          Inconvenientes: Cualquier procesado posterior del vector
197:          debe tratar las componentes marcadas de una forma especial
198:
199:       - Físico
200:          Implica desplazar 1 posición a la izquierda, todas las componentes que hay a la derecha de
201:          la que queremos borrar.
202:
203:          Tiene justo las ventajas e incovenientes contrarias que el método anterior.
204:
205:        En esta versión, implementamos el borrado físico.
206:    */
207:
208:    // Elimina una componente, dada por su posición
209:    void Elimina (int posicion){
210:       /*
211:       Algoritmo:
212:
213:          Recorremos de izquierda a derecha toda las componentes
214:          que hay a la derecha de la posición a eliminar
215:             Le asignamos a cada componente la que hay a su derecha
216:       */
217:       if (posicion >= 0 && posicion < util){
218:          int tope = util-1;
219:
220:          for (int i = posicion ; i < tope ; i++)
221:             v[i] = v[i+1];
222:
223:          util--;
224:       }
225:    }
226:
227:    // Algoritmos de ordenación
228:
229:    void Ordena_por_Seleccion(){
230:       int pos_min;
231:
232:       for (int izda = 0 ; izda < util ; izda++){
233:          pos_min = PosMinimoEntre(izda, util - 1);
234:          IntercambiaComponentesDirectamente(izda, pos_min);
235:       }
236:    }
237:
238:    void Ordena_por_Insercion(){
239:       int izda, i;
240:       char a_desplazar;
241:
242:       for (izda=1; izda < util; izda++){
243:          a_desplazar = v[izda];
244:
245:          for (i=izda; i > 0 && a_desplazar < v[i-1]; i--)
246:             v[i] = v[i-1];
247:
248:          v[i] = a_desplazar;
249:       }
250:    }
251:
252:    void InsertaOrdenadamente(char nuevo){
253:         int i;
254:
255:         if (util > TAMANIO){
256:            for (i=util; i>0 && nuevo < v[i-1]; i--)
257:                 v[i] = v[i-1];
258:
```

```
259:            v[i] = nuevo;
260:            util++;
261:         }
262:    }
263:
264:
265:    void Ordena_por_Burbuja(){
266:       int izda, i;
267:
268:       for (izda = 0; izda < util; izda++)
269:         for (i = util-1 ; i > izda ; i--)
270:           if (v[i] < v[i-1])
271:              IntercambiaComponentesDirectamente(i, i-1);
272:    }
273:
274:    void Ordena_por_BurbujaMejorado(){
275:       int izda, i;
276:       bool cambio;
277:
278:       cambio= true;
279:
280:       for (izda=0; izda < util && cambio; izda++){
281:         cambio=false;
282:
283:         for (i=util-1 ; i>izda ; i--)
284:           if (v[i] < v[i-1]){
285:              IntercambiaComponentesDirectamente(i, i-1);
286:              cambio=true;
287:           }
288:       }
289:    }
290:
291:    void AniadeVarios(SecuenciaCaracteres nuevos){
292:       int totales_a_aniadir = nuevos.Utilizados();
293:
294:       for (int i = 0; i < totales_a_aniadir; i++)
295:          Aniade(nuevos.Elemento(i)); // Es importante entender
296:    }
297:
298:
299:    SecuenciaCaracteres ToUpper(){
300:       SecuenciaCaracteres en_mayuscula;
301:
302:       for(int i = 0; i < util; i++)
303:          en_mayuscula.Aniade(toupper(v[i]));
304:
305:       return en_mayuscula;
306:    }
307:    bool EsVocal(int indice){
308:    /*
309:    Nos vale en privado y en publico ya que el objeto puede/debe contener
310:    el metodo tanto para acceder desde el main como para calculos dentro
311:    de otros metodos privados.
312:    */
313:    }
314:
315:    bool EsVocal(char caracter){
316:    /*
317:    No tiene sentido meterlo en la clase como método publico porque llegaremos
318:    a conclusiones absurdas como bool EsVocal('e'). ¿Es false porque no existe
319:    en cadena = "Adios" pero es true porque 'e' es una vocal?. No tiene sentido
320:    Sin embargo, si nos vale como privado para hacer calculos o como funcion
321:    global para otras cadenas o vectores.
322:    */
323:    }
324: };
325:
326:
327: int main(){
328:    SecuenciaCaracteres cadena;
329:    int pos_car;
330:    char car;
331:    bool es_vocal = false;
332:
333:    car = cin.get();
334:    while(car != '\n'){     //Pide caracteres hasta introducir 'enter'
335:       cadena.Aniade(car);
336:       car = cin.get();
337:    }
338:    cout << "Introduce la posicion del caracter que quieres comprobar si es vocal o no: ";
339:    cin >> pos_car;
340:
341:    cadena.EsVocal(pos_car);
342:
343: }
```

```
1: ////////////////////////////////////////////////////////////////////////
2: //
3: // Fundamentos de Programación
4: // ETS Informática y Telecomunicaciones
5: // Universidad de Granada
6: // Autor: Don Oreo
7: //
8: ////////////////////////////////////////////////////////////////////////
9:
10: // Mapa de distancias entre ciudades
11:
12: #include <iostream>
13: #include <cmath>
14: using namespace std;
15:
16:
17:
18: // ------------------------------------------------------------------------
19: class MapaDistancias{
20: private:
21:     static const int NUM_MAX_CIUDADES = 50;
22:     static const int DISTANCIA_NULA = 0;
23:     double mat_dist[NUM_MAX_CIUDADES][NUM_MAX_CIUDADES];
24:     int  num_ciudades;
25:
26:     bool EsCorrectaDistancia(double distancia){
27:        return distancia > DISTANCIA_NULA;
28:     }
29:
30:     bool EsCorrectoIndice(int indice){
31:       return 0 <= indice && indice < num_ciudades;
32:     }
33:
34:     bool EsCorrectaPosicion(int origen, int destino){
35:        return  EsCorrectoIndice(origen) && EsCorrectoIndice(destino);
36:     }
37:
38: public:
39:    MapaDistancias (int numero_ciudades)
40:       : num_ciudades(numero_ciudades)
41:    {
42:          for(int i = 0; i < NUM_MAX_CIUDADES; i++){
43:             for( int j = 0; j < NUM_MAX_CIUDADES; j++)
44:                mat_dist[i][j] = DISTANCIA_NULA;
45:          }
46:    }
47:
48:    int Capacidad(){
49:       return NUM_MAX_CIUDADES;
50:    }
51:
52:    int NumCiudades(){
53:       return num_ciudades;
54:    }
55:    double DistanciaCiudad(int pos_i, int pos_j){
56:       return mat_dist[pos_i][pos_j];
57:    }
58:
59:    void ModificaDistancia(int una, int otra, double distancia){
60:       if (EsCorrectaDistancia(distancia) && EsCorrectaPosicion(una, otra)){
61:          mat_dist[una][otra] = mat_dist[otra][una] = distancia;
62:       }
63:    }
64:
65:    int CiudadMejorConectada(){
66:       int indice_mas_conectada = -1;
67:       int max_conex = -1, num_conex;
68:
69:       for (int origen = 0; origen < num_ciudades; origen++){
70:          num_conex = 0;
71:
72:          for (int destino = 0; destino < num_ciudades; destino++)
73:             if (mat_dist[origen][destino] != 0)
74:                num_conex++;
75:
76:          if (num_conex > max_conex){
77:             max_conex = num_conex;
78:             indice_mas_conectada = origen;
79:          }
80:       }
81:
82:       return indice_mas_conectada;
83:    }
84:
85:    int MejorEscalaEntre (int origen, int destino){
86:       int escala_de_min_distancia = -1;
```

```
87:       double distancia_con_escala;
88:       double min_distancia = INFINITY;
89:
90:       for (int escala = 0; escala < num_ciudades; escala++){
91:          distancia_con_escala = 0;
92:
93:          if (mat_dist[origen][escala] != 0 && mat_dist[escala][destino] != 0)
94:             distancia_con_escala = mat_dist[origen][escala] +
95:                                    mat_dist[escala][destino];
96:
97:          if (distancia_con_escala != 0){
98:             if (distancia_con_escala < min_distancia){
99:                escala_de_min_distancia = escala;
100:               min_distancia = distancia_con_escala;
101:            }
102:         }
103:      }
104:
105:      return escala_de_min_distancia;
106:   }
107:
108:   string ToString() {
109:      string cadena;
110:
111:      for (int i = 0; i < num_ciudades; i++){
112:         for (int j = 0; j < num_ciudades; j++){
113:            cadena.append(to_string(mat_dist[i][j]));
114:            cadena.append("\t");
115:         }
116:         cadena.append("\n");
117:      }
118:      return cadena;
119:   }
120: };
121:
122:
123:
124: // ------------------------------------------------------------------------
125: // ------------------------------------------------------------------------
126: int main (){
127:    int ciudad_mas_conectada;
128:    int origen, destino, escala;
129:    int num_ciudades;
130:    const int TERMINADOR_CIUDADES = -1;
131:
132:    cout << "Mapa de distancias"
133:         << "\n\nIntroduzca los datos en el siguiente orden:"
134:         << "\na) Número de ciudades"
135:         << "\nb) Distancias entre ellas en forma de matriz diagonal superior"
136:         << "\n   Lista de índices de ciudades para las que se quiere ver "
137:         << "\n   si están todas conectadas entre sí. Terminador: "
138:         << TERMINADOR_CIUDADES
139:         << "\n"
140:         << "\n   Ciudad de origen y ciudad de destino."
141:         << "\n\n";
142:
143:    cin >> num_ciudades;
144:
145:    MapaDistancias mapa(num_ciudades);
146:
147:    for (int i = 0; i < num_ciudades - 1; i++){
148:       for (int j = i+1; j < num_ciudades; j++){
149:          double distancia;
150:
151:          cin >> distancia;
152:          mapa.ModificaDistancia(i, j, distancia);
153:       }
154:    }
155:
156:    /*for(int i = 0; i < num_ciudades; i++){
157:       cout << "\n";
158:       for( int j = 0; j < num_ciudades; j++)
159:          cout << mapa.DistanciaCiudad(i,j) << "\t";
160:    }*/
161:
162:    cin >> origen >> destino;
163:
164:    //-------------------------------------------------------------
165:    // Ciudad mejor conectada (con mayor número de conexiones directas)
166:    ciudad_mas_conectada = mapa.CiudadMejorConectada();
167:
168:    cout << "La ciudad con más conexiones directas es la ciudad: "
169:         << ciudad_mas_conectada;
170:
171:
172:    //-------------------------------------------------------------
```

```
173:    // Mejor escala entre origen y destino
174:    escala = mapa.MejorEscalaEntre(origen,destino);
175:
176:    if (escala == -1)
177:       cout << "No existe escala" << endl;
178:    else
179:       cout << "\nLa mejor escala entre " << origen << " y "
180:            << destino << " es " << escala << endl;
181:    return 0;
182: }
183:
184: /*
185: Entrada:
186:
187:    5
188:    50  100   0    150
189:        70    0    0
190:              60   80
191:                   90
192:    0  4
193:
194: Salida:
195:
196:    La ciudad con más conexiones directas es la ciudad: 2
197:    La mejor escala entre 0 y 4 es 2
198: */
```

```cpp
////////////////////////////////////////////////////////////////////////
//
// Fundamentos de Programación
// ETS Informática y Telecomunicaciones
// Universidad de Granada
// Autor: Don Oreo
//
////////////////////////////////////////////////////////////////////////

// Túnel

#include <iostream>
#include <cmath>
#include "IV_FormateadorDoubles.cpp"
#include "IV_Instante.cpp"

using namespace std;


/*
Túnel en un momento dado:

matr:     ["4733MTI"  ,  "5232LTL"  , "3330PRB" ,   ?  , ... ,  ?  ]
seg_ent:  [    13     ,     79      ,    160    ,   ?  , ... ,  ?  ]
seg_sal:  [    96     ,    NULO     ,    NULO   , NULO , ... , NULO]

tot = 3
*/

class Tunel{
private:
   double distancia_km;

   static const int MAX_NUM_VEHICULOS = 100;
   string matriculas[MAX_NUM_VEHICULOS];
   int entradas[MAX_NUM_VEHICULOS];
   int salidas[MAX_NUM_VEHICULOS];
   int total_entradas = 0;

   static const int INSTANTE_NULO = -1;

public:
   Tunel(double long_tunel)
   :distancia_km(long_tunel)
   {
      for(int i = 0; i < MAX_NUM_VEHICULOS; i++)
         salidas[i] = -1;
   }

   string Matriculas(int posicion){
      return matriculas[posicion];
   }

   int Entradas(int posicion){
      return entradas[posicion];
   }

   int Salidas(int posicion){
      return salidas[posicion];
   }

   double Longitud(){
      return distancia_km;
   }

   int TotalEntradas(){
      return total_entradas;
   }

   void Entra(string matricula, int hora, int minuto, int segundo){
      int pos = -1;
      Instante inst_entrada(hora,minuto,segundo);
      int seg_totales = inst_entrada.SegundosTotales();

      for(int i = 0; i < total_entradas; i++)
         if(matricula == matriculas[i])
            pos = i;
      if(pos == -1){
         matriculas[total_entradas] = {matricula};
         entradas[total_entradas] = seg_totales;
         total_entradas++;
      }
      else{
         entradas[pos] = seg_totales;
         salidas[pos] = INSTANTE_NULO;
      }
```

```
87:     }
88:
89:     void Sale(string matricula, int hora, int minuto, int segundo){
90:        int pos = -1;
91:        Instante inst_salida(hora,minuto,segundo);
92:        int seg_totales = inst_salida.SegundosTotales();
93:
94:        for(int i = 0; i < total_entradas; i++)
95:           if(matricula == matriculas[i])
96:              pos = i;
97:
98:        if(pos != -1)
99:           salidas[pos] = seg_totales;
100:
101:    }
102:
103:    bool HaSalido(int pos){
104:       bool ha_salido = true;
105:
106:       if(Salidas(pos) == -1)
107:          ha_salido = false;
108:
109:       return ha_salido;
110:    }
111:
112:    double Velocidad(int pos){
113:       double velocidad;
114:       double diferencia_hora = (Salidas(pos) - Entradas(pos))/3600.0;
115:
116:       velocidad = Redondea(distancia_km/diferencia_hora,1);
117:
118:       return velocidad;
119:    }
120:
121: };
122:
123:
124: int main(){
125:    const char FIN_ENTRADA = '#';
126:    const char ENTRADA = 'E';
127:    const char SALIDA  = 'S';
128:    char acceso;
129:    bool error_lectura;
130:
131:    string matricula;
132:    string cadena;
133:    double long_tunel;
134:    int hora, min, seg;
135:
136:    cin >> long_tunel;
137:    Tunel tunel(long_tunel);
138:
139:    //Escribe E para entrada, S para salida y # para terminar la lectura
140:    cin >> acceso;
141:    error_lectura = false;
142:
143:    while (acceso != FIN_ENTRADA && !error_lectura){
144:       cin >> matricula;
145:       cin >> hora >> min >> seg;
146:
147:       if (acceso == ENTRADA)
148:          tunel.Entra(matricula,hora,min,seg);
149:       else if (acceso == SALIDA)
150:          tunel.Sale(matricula,hora,min,seg);
151:       else
152:          error_lectura = true;
153:
154:       cin >> acceso;
155:    }
156:
157:    //-------------------------------------------------------------
158:
159:    FormateadorDoubles format_veloc("," km/h", 1);
160:    if (error_lectura)
161:       cout << "\nSe produjo un error en la lectura. " << endl;
162:    else{
163:       int total_entradas = tunel.TotalEntradas();
164:       for (int i = 0; i < total_entradas; i++){
165:          cadena += "\n\nMatricula:\t" + tunel.Matriculas(i) +
166:          "\nVelocidad:\t";
167:
168:          if(!tunel.HaSalido(i))
169:             cadena += "No ha salido";
170:          else
171:             cadena += format_veloc.GetCadena(tunel.Velocidad(i));
172:       }
```

```
173:    }
174:
175:    cout << cadena << endl;
176: }
177:
178:    // longitud_túnel  <entrada_o_salida  Matrícula# Instante> ... #
179:
180: // Entrada:
181: /*
182: 3.4
183: E 4733MTI 0 0 13
184: E 5232LTL 0 1 19
185: S 4733MTI 0 1 36
186: E 3330PRB 0 2 40
187: S 5232LTL 0 3 25
188: #
189: */
190:
191: // Salida:
192: /*
193: Matrícula:      4733MTI
194: Velocidad:      147.5 km/h
195:
196: Matrícula:      5232LTL
197: Velocidad:      97.1 km/h
198:
199: Matrícula:      3330PRB
200: Velocidad:      No ha salido
201: */
202:
203:
204: ////////////////////////////////////////////////////////////////
205:
206: // Entrada:
207: /*
208:    3.4
209: E 4733MTI 0 0 13
210: E 1976KEX 0 0 34
211: E 7717UQS 0 0 47
212: E 4744SEU 0 0 56
213: E 5232LTL 0 1 19
214: S 4733MTI 0 1 36
215: E 6188MOH 0 1 36
216: E 6603JHQ 0 2 4
217: E 6898DVW 0 2 17
218: E 3330PRB 0 2 40
219: S 1976KEX 0 2 53
220: E 1758HRV 0 2 56
221: E 8210YVI 0 3 9
222: S 5232LTL 0 3 25
223: S 6603JHQ 0 3 25
224: S 7717UQS 0 3 29
225: S 6188MOH 0 3 29
226: E 9265JJA 0 3 35
227: S 4744SEU 0 3 40
228: E 4864DUN 0 3 49
229: S 3330PRB 0 3 51
230: E 1071VVF 0 3 54
231: S 1758HRV 0 4 30
232: E 5917FBY 0 4 43
233:    */
234:
235:    // Salida:
236:    /*
237:    Matrícula:   4733MTI
238: Velocidad:      147.5 km/h
239:
240: Matrícula:      1976KEX
241: Velocidad:      88.1 km/h
242:
243: Matrícula:      7717UQS
244: Velocidad:      75.6 km/h
245:
246: Matrícula:      4744SEU
247: Velocidad:      74.6 km/h
248:
249: Matrícula:      5232LTL
250: Velocidad:      97.1 km/h
251:
252: Matrícula:      6188MOH
253: Velocidad:      108.3 km/h
254:
255: Matrícula:      6603JHQ
256: Velocidad:      151.1 km/h
257:
258: Matrícula:      6898DVW
```

```
259: Velocidad:      No ha salido
260:
261: Matrícula:      3330PRB
262: Velocidad:      172.4 km/h
263:
264: Matrícula:      1758HRV
265: Velocidad:      130.2 km/h
266:
267: Matrícula:      8210YVI
268: Velocidad:      No ha salido
269:
270: Matrícula:      9265JJA
271: Velocidad:      No ha salido
272:
273: Matrícula:      4864DUN
274: Velocidad:      No ha salido
275:
276: Matrícula:      1071VVF
277: Velocidad:      No ha salido
278:
279: Matrícula:      5917FBY
280: Velocidad:      No ha salido
281:    */
282:
```