Programación 2

Examen de teoría (julio 2015)

8 de julio de 2015, 14:30

Instrucciones

- **Duración: 3 horas**
- El fichero del primer problema debe llamarse `mantenimiento.cc`. Para el segundo problema es necesario entregar cuatro ficheros, llamados `Planet.cc, Planet.h, Route.cc, Route.h`. Pon tu DNI y tu nombre en un comentario al principio de los ficheros fuente.
- La entrega se realizará como en las prácticas, a través del servidor del DLSI (`http://pracdlsi.dlsi.ua.es`), en el enlace `Programación 2`. Puedes realizar varias entregas, aunque sólo se corregirá la última.
- En la página web de la asignatura `http://www.dlsi.ua.es/asignaturas/p2` tienes disponibles algunos ficheros que te pueden servir de ayuda para resolver los problemas del examen, y el apartado `Reference` de la web `www.cplusplus.com`.

Problemas

1. **(5 puntos)**

   Tras la celebración de un macroconcierto, el servicio de mantenimiento del Albergue Juvenil de la ciudad detecta un cierto deterioro en algunas instalaciones. Para poder organizar las tareas de restauración deciden ir escribiendo en un fichero los productos que deben ser repuestos a medida que van realizando la inspección.

   Por otra parte se dispone, también en un fichero de texto, de un catálago con productos y precios (un número entero), en el que en cada línea aparece el nombre del producto seguido del precio.

   Se pide realizar un programa, llamado `mantenimiento`, que contabilice el total de productos que se deben comprar y el coste total de los productos, y que genere un fichero binario con los datos de la compra. Se puede suponer que en el fichero del catálogo los productos no están repetidos y que aparecen todos los productos necesarios. En el fichero binario "`compra.dat`" se escribirá un registro del siguiente tipo por cada producto que sea necesario comprar:

```
const int MAXNOM=10;

typedef struct {
  char nombre[MAXNOM];
  int precio;
  int unidades;
} ProductoBin;
```

   Si el nombre de un producto no cabe en el espacio reservado en dicho registro, se debe recortar adecuadamente para que quepa (tal y como se ha explicado en clase de teoría).

   El programa debe ser llamado mediante[1]

```
./mantenimiento productos.txt catalogo.txt
```

[1] Tanto si el programa no recibe ambos parámetros como si alguno de los archivos no se puede abrir, debe mostrar el error correspondiente y finalizar.

Un ejemplo de ficheros y la salida correspondiente podría ser:

productos.txt

```
sabanas espejo sabanas cenicero cenicero cenicero toalla_mediana
toalla_grande toalla_mediana sabanas sabanas toalla_grande espejo
sabanas cenicero toalla_mediana
```

catalogo.txt

```
sabanas 15
toallita 3
toalla_grande 7
toalla_mediana 5
espejo 14
television 300
cenicero 1
```

Salida:

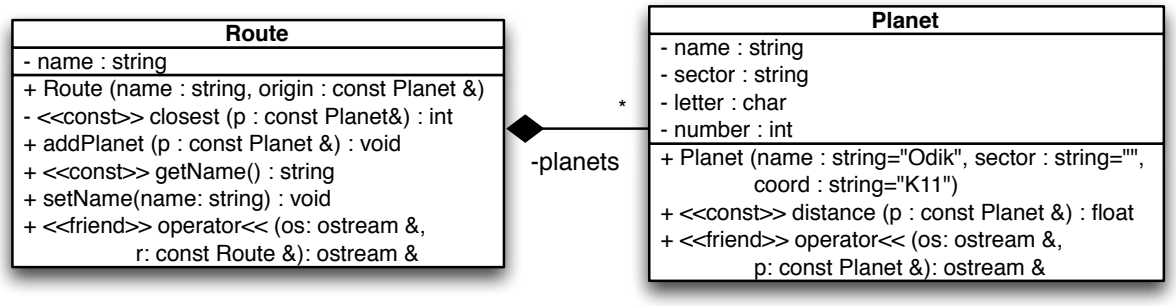
```
sabanas: 5 x 15 = 75
espejo: 2 x 14 = 28
cenicero: 4 x 1 = 4
toalla_mediana: 3 x 5 = 15
toalla_grande: 2 x 7 = 14
------
Coste: 136
------
```

compra.dat (datos que se guardarían, recuerda que es un fichero binario)

```
sabanas   15 5
espejo    14 2
cenicero   1 4
toalla_me  5 3
toalla_gr  7 2
```

2. **(5 puntos)**

El Halcón Milenario debe hacer una ruta por varios planetas para entregar unos paquetes.



Un planeta (`Planet`) contiene un nombre, el sector de la galaxia donde se encuentra, y unas coordenadas galácticas codificadas con una letra en mayúscula seguida por un número, por ejemplo `K17`. El constructor debe guardar estos datos, separando la coordenada recibida en la letra y el número[2]. El operador salida imprimirá el planeta con el formato que puede verse en el ejemplo de ejecución (nombre, sector entre paréntesis y coordenada). Esta clase también tiene un método `distance` para calcular la distancia interestelar entre dos planetas, que viene dada por la siguiente fórmula[3]:

$$distance(a, b) = 100 * abs(a.letter - b.letter) + abs(a.number - b.number) \quad (1)$$

La clase `Route` calcula la ruta a seguir a partir de un planeta de origen. Para ello, el constructor debe asignar el nombre y añadir el planeta recibido (el de origen) al vector de planetas. También tenemos un método `closest`, que calcula el planeta más cercano al que recibe como parámetro y devuelve su posición en el vector de planetas. Si el vector está vacío, debe devolver `-1`. El método `addPlanet` añadirá un nuevo planeta al vector de forma que la ruta esté ordenada en todo momento. Para ello buscará el planeta más cercano llamando al método `closest`, y almacenará el nuevo planeta justo a continuación de su planeta más cercano[4].

Finalmente, el operador salida imprimirá la lista de planetas en el orden de la ruta, que es el que deben tener en el vector. Dado el siguiente `main.cc`:

```
#include "Route.h"

int main()
{
  Route r("My route", Planet("Endor","Zuma","H16"));

  r.addPlanet(Planet("Felucia","Thanium","R6"));
  r.addPlanet(Planet("Geonosis","Arkanis","R16"));
  r.addPlanet(Planet("Ansion","Churnis","I6"));
  r.addPlanet(Planet("Phoebus","Thanium","R6"));
  r.addPlanet(Planet("Subterrel","Subterrel","L20"));

  cout <<  r << endl;
}
```

el programa debería imprimir:

```
My route
----
Endor (Zuma), H16
Ansion (Churnis), I6
Subterrel (Subterrel), L20
Felucia (Thanium), R6
Phoebus (Thanium), R6
Geonosis (Arkanis), R16
```

[2]Se asumirá que la coordenada es siempre correcta, y que contiene una letra y un número de cualquier longitud
[3]La función `abs` (en la librería `cstdlib`) devuelve el valor absoluto del número entero que recibe
[4]La idea es que en todo momento vayamos al planeta más cercano, lo cual no es óptimo pero permite simplificar el problema