Facultad de Informática – Universidad Complutense de Madrid

Fundamentos de la programación II – grupos DG-ADE, D, G, I

Curso 2021-2022

Examen convocatoria ordinaria

EJERCICIO1 [3 PUNTOS]

Tenemos definido un tipo `tMatriz` para matrices cuadradas de dimensión máxima `MAX=10`

```
struct tMatriz {
    int matriz[MAX][MAX];
    int dimension; // nº de filas y columnas usadas
};
```

Tu trabajo consiste en desarrollar los siguientes subprogramas para determinar si una matriz `m` de tipo `tMatriz` cumple ciertas características e invocarlos desde el programa principal:

- ✓ **[1,25 puntos]** Desarrollar e invocar el subprograma

  ```
  bool esXpar(const tMatrizCuadrada& m);
  ```

  que comprueba si la matriz `m` es *X-par*. Decimos que una matriz de tipo `tMatriz` es *X-par* si la suma de los valores de la diagonal principal y de la diagonal secundaria es un número par.
  Por ejemplo: la matriz 1 es X-par (la suma de los valores de ambas diagonales es 20) pero la matriz 2 no es X-par (dicha suma es 21) (se marcan en rojo los valores de las diagonales)

Matriz 1	Matriz 2
1 2 3 4 5 5 1 2 3 4 4 5 1 2 3 3 4 5 1 2 2 3 4 5 1	1 2 3 4 5 5 1 2 3 4 4 5 1 2 3 3 5 5 1 2 2 3 5 6 1

- ✓ **[1,75 puntos]** Desarrollar e invocar el subprograma

  ```
  void esCuadradoRotado(const tMatrizCuadrada& m, bool& loEs);
  ```

  que comprueba si la matriz `m` es un *cuadrado rotado*. Decimos que una matriz de tipo `tMatriz` es un *cuadrado rotado* si cada fila (a excepción de la primera) contiene los elementos de la fila anterior rotados 1 posición a la derecha.
  Por ejemplo: la matriz 1 es un cuadrado rotado pero la matriz 2 no lo es (en la matriz 2 se marcan en rojo los valores que la hacen no cumplir la condición).

Matriz 1	Matriz 2
1 2 3 4 5 5 1 2 3 4 4 5 1 2 3 3 4 5 1 2 2 3 4 5 1	1 2 3 4 5 5 1 2 3 4 4 5 1 2 3 3 5 5 1 2 2 3 5 6 1

EJERCICIO2 [7 PUNTOS]

Una empresa de transporte urgente de paquetería dispone de 10 pequeñas furgonetas que por la mañana son cargadas con paquetes rumbo a su destino. Cada furgoneta se encarga del reparto en un código postal diferente y se puede cargar con un número máximo de 8 paquetes. Los paquetes, por su

parte, se caracterizan por un identificador y el código postal destino, y esperan en fila cada mañana a ser cargados en alguna furgoneta y ser trasladados a su destino. Los paquetes que no pueden cargarse en las furgonetas pueden ser recogidos en mano por sus destinatarios.

Para representar la información se ha pensado en los siguientes tipos y módulos:

- Módulo `Paquetes`. En él se definen los siguientes tipos:
  - `tPaquete`, que representa un paquete: tipo `struct` compuesto de identificador –`string` compuesto por una única palabra–, código postal –entero– e indicador de si ha sido cargado o no en una furgoneta –`bool`. (*Este tipo ya está definido en la plantilla proporcionada como parte del material del examen*.)
  - `tListaPaquetes`, que representa una lista de tamaño variable de hasta 50 paquetes que puede repartir la empresa cada día. Se trata de una lista soportada por un array de punteros a paquetes de tipo `tPaquete`.
- Módulo `Furgonetas`. En él se definen los siguientes tipos:
  - `tCargados`, que representa una lista de tamaño variable de hasta 8 identificadores de paquetes soportada por un array dinámico.
  - `tFurgoneta`, que representa una furgoneta: tipo `struct` compuesto por código postal en el que la furgoneta realiza el reparto –entero– y una lista de tipo `tCargados` con los identificadores de los paquetes que se cargan la furgoneta.
  - `tListaFurgonetas`, un array estático de 10 furgonetas de tipo `tFurgoneta`.

Tras declarar los tipos `tListaPaquetes`, `tCargados`, `tFurgoneta` y `tListaFurgonetas` en los módulos correspondientes **[1 PUNTO]**, tienes que desarrollar un pequeño prototipo que sea capaz de llevar a cabo las siguientes acciones en el orden indicado:

1. **[1 PUNTO]** Cargar los paquetes en una lista de tipo `tListaPaquetes` desde el archivo `paquetes.txt`. Para ello implementarás el subprograma
   ```
   bool cargarPaquetes(tListaPaquetes& listaPaquetes);
   ```
   El archivo `paquetes.txt` contiene una primera línea con el número de paquetes (<= 50) y luego una línea por cada paquete conteniendo el identificador seguido del código postal.
2. **[0,25 PUNTOS]** Mostrar la lista de paquetes cargada, para lo que implementarás el subprograma
   ```
   void mostrarPaquetes(const tListaPaquetes& listaPaquetes);
   ```
3. **[1 PUNTO]** Cargar los códigos postales en los que repartirán las furgonetas. Se carga desde el archivo `codigos.txt` a una lista de tipo `tListaFurgonetas`. Además, cada furgoneta debe quedar con su lista de tipo `tCargados` vacía.
   Para esta tarea implementarás el subprograma
   ```
   bool cargarCodigos(tListaFurgonetas listaFurgonetas);
   ```
   El fichero `codigos.txt` tiene en la primera línea el código postal en el que repartirá la primera furgoneta de la lista, en la segunda línea tiene el código postal en el que repartirá la segunda furgoneta, y así sucesivamente. Los códigos postales no se repiten y aparecen en orden creciente en el fichero, con lo que la lista quedará ordenada crecientemente por código postal.
4. **[0,5 PUNTOS]** Mostrar el contenido de la lista de furgonetas, para lo que implementarás el subprograma
   ```
   void mostrarFurgonetas(const tListaFurgonetas listaFurgonetas);
   ```
5. **[1,5 PUNTOS]** Cargar paquetes en las furgonetas, colocando todos los paquetes que sea posibles en las furgonetas. Si aún hay sitio para un paquete en una furgoneta que reparta en el código postal destino del paquete, el identificador del paquete se guardará en la lista de identificadores de paquetes de la furgoneta y el paquete se marcara como cargado. Si no hay

furgoneta que reparta en el código postal destino del paquete o ya no hay sito en la furgoneta que reparte en dicho código postal, el paquete se quedará sin cargar.
Para implementar esta funcionalidad, desarrollarás el subprograma

```
void cargarPaquetes(tListaFurgonetas listaFurgonetas, tListaPaquetes& listaPaquetes);
```

Y para saber si hay una furgoneta que reparta en un cierto código postal desarrollarás el subprograma

```
int buscarFurgoneta(const tListaFurgonetas listaFurgonetas, int codigo);
```

que implementará una búsqueda binaria recursiva (devuelve -1 si no se encuentra la furgoneta; puedes incorporar algún parámetro más si lo consideras necesario).

6. Volver a mostrar la lista de paquetes, usando el subprograma `mostrarPaquetes` del apartado 2, y el contenido de la lista de furgonetas, usando el subprograma `mostarFurgonetas` del apartado 4, para comprobar los resultados de la carga de paquetes en furgonetas.
7. **[1,25 PUNTOS]** Recoger un paquete que no haya sido cargado. Se introduce por teclado el identificador del paquete y, si existe un paquete con ese identificador y no ha sido cargado en una furgoneta, se elimina de la lista de paquetes para que el destinatario se lo pueda llevar a casa (se debe preservar la disposición relativa de los paquetes que permanecen en la lista). Si el paquete no existe o ya ha sido cargado en alguna furgoneta, la acción no surtirá ningún efecto. Para hacer todo esto implementarás, al menos, el subprograma

```
void recogerPaquete(tListaPaquetes& listaPaquetes);
```

8. Volver a mostrar la lista de paquetes para comprobar el efecto de la recogida, usando de nuevo para ello el subprograma `mostrarPaquetes` del apartado 2.
9. **[0,5 PUNTOS]** El programa deberá terminar sin generar fugas de memoria, para lo cual deberás implementar los subprogramas `void liberarFurgonetas(tListaFurgonetas listaFurgonetas);` y `void liberarPaquetes(tListaPaquetes& listaPaquetes);` que liberan la memoria dinámica usada por la lista de furgonetas y la lista de paquetes, respectivamente.

En la siguiente página, puedes ver el comportamiento del prototipo para los ficheros `paquetes.txt` y `codigos.txt` proporcionados como material de examen.

- LISTA DE PAQUETES AL COMIENZO -

id01 va a 28100 - cargado: NO
id02 va a 28200 - cargado: NO
id03 va a 28100 - cargado: NO
id04 va a 28100 - cargado: NO
id05 va a 28100 - cargado: NO
id06 va a 28100 - cargado: NO
id07 va a 28100 - cargado: NO
id08 va a 28100 - cargado: NO
id09 va a 28100 - cargado: NO
id10 va a 28100 - cargado: NO
id11 va a 28200 - cargado: NO
id12 va a 28200 - cargado: NO
id13 va a 28500 - cargado: NO
id14 va a 28000 - cargado: NO

Visualización de la lista de paquetes con `mostrarPaquetes` tras ser cargada desde `paquetes.txt`

- LISTA DE FURGONETAS AL COMIENZO -

Furgoneta 1 reparte en 28000 - Sin paquetes asignados
Furgoneta 2 reparte en 28100 - Sin paquetes asignados
Furgoneta 3 reparte en 28200 - Sin paquetes asignados
Furgoneta 4 reparte en 28300 - Sin paquetes asignados
Furgoneta 5 reparte en 28400 - Sin paquetes asignados
Furgoneta 6 reparte en 28600 - Sin paquetes asignados
Furgoneta 7 reparte en 28700 - Sin paquetes asignados
Furgoneta 8 reparte en 28800 - Sin paquetes asignados
Furgoneta 9 reparte en 28900 - Sin paquetes asignados
Furgoneta 10 reparte en 28990 - Sin paquetes asignados

Visualización de la lista de furgonetas con `mostrarFurgonetas` después de cargar los códigos desde `códigos.txt`

- LISTA DE PAQUETES TRAS LA CARGA EN FURGONETAS -

id01 va a 28100 - cargado: SI
id02 va a 28200 - cargado: SI
id03 va a 28100 - cargado: SI
id04 va a 28100 - cargado: SI
id05 va a 28100 - cargado: SI
id06 va a 28100 - cargado: SI
id07 va a 28100 - cargado: SI
id08 va a 28100 - cargado: SI
id09 va a 28100 - cargado: SI
id10 va a 28100 - cargado: NO
id11 va a 28200 - cargado: SI
id12 va a 28200 - cargado: SI
id13 va a 28500 - cargado: NO
id14 va a 28000 - cargado: SI

Visualización de la lista de paquetes con `mostrarPaquetes` tras ser cargados los paquetes en furgonetas con `cargarPaquetes`

(el paquete id10 no se puede cargar porque ya no hay sitio en la furgoneta 2 –la que reparte en el código postal 28100– y el paquete id13 no se puede cargar porque no hay furgoneta que reparta en el código postal 28500)

- LISTA DE FURGONETAS TRAS LA CARGA DE PAQUETES -

Furgoneta 1 reparte en 28000 - Paquetes asignados: id14

Furgoneta 2 reparte en 28100 - Paquetes asignados: id01 id03 id04 id05 id06 id07 id08 id09

Furgoneta 3 reparte en 28200 - Paquetes asignados: id02 id11 id12

Furgoneta 4 reparte en 28300 - Sin paquetes asignados

Furgoneta 5 reparte en 28400 - Sin paquetes asignados

Furgoneta 6 reparte en 28600 - Sin paquetes asignados

Furgoneta 7 reparte en 28700 - Sin paquetes asignados

Furgoneta 8 reparte en 28800 - Sin paquetes asignados

Furgoneta 9 reparte en 28900 - Sin paquetes asignados

Furgoneta 10 reparte en 28990 - Sin paquetes asignados

Visualización de la lista de furgonetas con `mostrarFurgonetas` tras ser cargados los paquetes en furgonetas con `cargarPaquetes`

… continúa a ejecución donde NO se recoge un paquete…

… continúa a ejecución donde SÍ se recoge un paquete…

Introduzca el identificador del paquete a recoger: id01

- LISTA DE PAQUETES TRAS LA RECOGIDA EN MANO -

id01 va a 28100 - cargado: SI

id02 va a 28200 - cargado: SI

id03 va a 28100 - cargado: SI

id04 va a 28100 - cargado: SI

id05 va a 28100 - cargado: SI

id06 va a 28100 - cargado: SI

id07 va a 28100 - cargado: SI

id08 va a 28100 - cargado: SI

id09 va a 28100 - cargado: SI

id10 va a 28100 - cargado: NO

id11 va a 28200 - cargado: SI

id12 va a 28200 - cargado: SI

id13 va a 28500 - cargado: NO

id14 va a 28000 - cargado: SI

Visualización de la lista de paquetes con `mostrarPaquetes` (no se puede recoger el paquete solicitado porque ya está cargado en una furgoneta y la lista queda intacta)

Introduzca el identificador del paquete a recoger: id10

- LISTA DE PAQUETES TRAS LA RECOGIDA EN MANO -

id01 va a 28100 - cargado: SI

id02 va a 28200 - cargado: SI

id03 va a 28100 - cargado: SI

id04 va a 28100 - cargado: SI

id05 va a 28100 - cargado: SI

id06 va a 28100 - cargado: SI

id07 va a 28100 - cargado: SI

id08 va a 28100 - cargado: SI

id09 va a 28100 - cargado: SI

id11 va a 28200 - cargado: SI

id12 va a 28200 - cargado: SI

id13 va a 28500 - cargado: NO

id14 va a 28000 - cargado: SI

Visualización de la lista de paquetes con `mostrarPaquetes` (se ha podido recoger el paquete solicitado y desaparece de la lista conservando la disposición relativa de los demás paquetes)