Examen 2021-06-15

Examen teórico

1 Introducción

El proceso de vacunación de la población general es una labor compleja y con muchas aristas, que involucra variables no controladas como problemas en el suministro de viales, problemas logísticos con los centros disponibles para la inoculación, problemas de inventario (necesidad de agujas hipodérmicas adecuadas) o incluso problemas por falta de personal sanitario cualificado para vacunar, entre otras cosas. El equipo del actual director del Centro de Coordinación de Alertas y Emergencias Sanitarias del Ministerio de Sanidad, Fernando Simón, nos ha encargado la tarea de ayudarles en el cálculo de previsiones de vacunación, asumiendo un flujo de vacunas por parte de los proveedores predeterminado.

En concreto, la información que nos facilitan desde el Ministerio de Sanidad son varios ficheros de texto con datos tabulados, que **usan el tabulador para separar cada uno de los campos de datos**. Estos ficheros son los siguientes:

- Ficheros `prevision_julio.tsv` y `prevision_agosto.tsv`, que contiene una previsión de las llegadas a cada comunidad autónoma de viales de las diferentes marcas de vacunas aprobadas por la Agencia Europea del Medicamento durante el mes de julio y de agosto, respectivamente. Las primeras líneas de estos ficheros son como las siguientes:

```
COD_CA  PF  MD  JJ  AZ
0   2500    2900    6500    1200
2   7500    6900    8500    1800
1   1500    2700    3500    600
    ```

- Fichero `vacunados.tsv`, que contiene el número de vacunados de cada comunidad autónoma en su cohorte (conjunto, serie) poblacional correspondiente. Las primeras líneas de este fichero son las siguientes:

```plaintext
COD_CA  0-17    18-29   30-39   40-49   50-59   60-69   70-79+
1   180 242 144 932 547 966 1520
0   510 422 112 232 477 566 1990
2   110 212 44  432 747 866 990
```

- Fichero `poblacion.tsv`, que contiene la población de cada comunidad autónoma en su cohorte correspondiente. Las primeras líneas de este fichero son las siguientes:

```
COD_CA  0-17    18-29   30-39   40-49   50-59   60-69   70-79+
2   14104   30224   43002   53437   52866   39920   56631
0   15217   12333   13112   40932   44377   15606   59090
1   26802   64230   13144   53202   30947   15066   65020
```

Según nos han indicado desde el Ministerio, `COD_CA` es un código numérico que identifica a cada comunidad autónoma. **Nos han destacado que las líneas contenidas en los ficheros de información no están ordenadas según este código ni ningún otro criterio** (como se puede ver en los ejemplos mostrados).

Para facilitar algunas de las tareas que se solicitan a continuación, los profesores de la asignatura hemos desarrollado la función `nombreDeCA`, que recibe un entero que representa la comunidad autónoma y devuelve una cadena, que es el nombre de la comunidad autónoma del entero dado por parámetro. En concreto, la cabecera de esta función es la siguiente:

```
// Devuelve el nombre de la comunidad autónoma representada
// por el entero cod_CA, dado por parámetro
string nombreDeCA(int cod_CA){
      ... // Función YA IMPLEMENTADA (la puedes utilizar, pero no debes
implementarla)
}
```

Por ejemplo, `nombreDeCA(0)` devuelve la cadena `"ANDALUCÍA"`, mientras que `nombreDeCA(1)` devuelve la cadena `"ARAGÓN"`.

También hemos definido los siguientes tipos de datos:

- El tipo de dato `PrevisionVacunasCA` permite almacenar la previsión de viales de cada vacuna, almacenando además en un campo de tipo entero el identificador de la comunidad autónoma a la que corresponde la previsión contenida en los otros campos. Este tipo de dato sirve para almacenar la información de los ficheros `prevision_julio.tsv` y `prevision_agosto.tsv`.

```
@datalcases.dataclass
class PrevisionVacunasCA:
    numPF: int
    numMD: int
    numJJ: int
    numAZ: int
    codigoCA: int
```

- El tipo de dato `PoblacionVacunadosCA` permite almacenar el número total de población y el número de vacunados de cada cohorte, almacenando también un campo (de tipo entero) para guardar el identificador de la comunidad autónoma correspondiente. Este tipo de dato sirve para almacenar la información de los ficheros `vacunados.tsv` y `poblacion.tsv`. La población total y la población vacunada se guardan en vectores, donde en la primera componente se encuentran los datos de la cohorte 0-17, en la segunda componente la cohorte 18-29, y así sucesivamente.

```
@datalcases.dataclass
class PoblacionVacunadosCA:
    vacunados: list[int]
    poblacion: list[int]
    codigoCA: int
```

Por último, también se ha desarrollado la función `distribuyeViales`, que se encarga de distribuir de manera equitativa la previsión de viales entre toda la población, por cada comunidad autónoma. Esta función recibe un vector de `PrevisionVacunasCA` y un vector `PoblacionVacunadosCA`, que representan la previsión de llegadas de viales y la población actual y vacunada hasta el momento de cada comunidad autónoma, respectivamente. La función devuelve un nuevo vector de `PoblacionVacunadosCA` que contiene la información actualizada con los nuevos vacunados de cada cohorte, según la previsión de llegada de viales dada por parámetro en el vector `vVac`. Su cabecera es la siguiente:

```
# Distribuye de manera equitativa la previsión de llegadas de viales entre
la población
def distribuyeViales(vVac, vPobl):
   # Función YA IMPLEMENTADA (la puedes utilizar, pero no debes implementarla)
    return
```

Para acabar de desarrollar lo solicitado desde el Ministerio, **es necesario que realices las siguientes tareas**:

2 Ejercicios

2.a Ejercicio 1: Desarrolla un módulo (procedimiento o función) `leePrevisiones` que se encargue de leer el fichero de previsiones y devolver un vector de tipo de dato `PrevisionVacunasCA` con la información leída

. El módulo recibirá como parámetro una cadena, que será el nombre del fichero que contiene la información a leer (fichero de previsiones de llegada de viales). Además, informará al usuario por pantalla con un mensaje de error (`"ERROR al abrir el fichero"`) en caso de que el fichero no se abra correctamente. **(1.50 puntos)**

```
// Lee el fichero de previsiones nombFich y devuelve
// un vector de PrevisionVacunasCA con la información dada por el fichero
??? leePrevisiones(string ??nombFich){
    // Por completar
}
```

```
// Lee el fichero de previsiones nombFich y devuelve
// un vector de PrevisionVacunasCA con la información dada por el fichero
vector<PrevisionVacunasCA> leePrevisiones(string nombFich){
    vector<PrevisionVacunasCA> v;
    ifstream f(nombFich);

    if(f.is_open()){
        // leemos cabecera
        string cad;
        getline(f, cad);
        // lectura de fichero
        while(!f.eof()){
            PrevisionVacunasCA elem;
```

```
            // leemos datos de prevision
            f >> elem.codigoCA;
            f >> elem.numPF;
            f >> elem.numMD;
            f >> elem.numJJ;
            f >> elem.numAZ;

            // lo añadimos
            v.push_back(elem);
        }
        // cerramos fichero
        f.close();
    }else{
        cout << "ERROR al abrir el fichero" << endl;
    }

    return v;
}
```

2.b Ejercicio 2: Desarrolla un módulo (procedimiento o función) `buscaCA` que se encargue de, dado un vector de `PoblacionVacunadosCA` y un código identificador de comunidad autónoma, en qué componente del vector se encuentra dicha comunidad autónoma

. Este módulo recibirá un vector `v` de `PoblacionVacunadosCA` y devolverá el índice del elemento del vector que contiene la comunidad autónoma identificada por `codigoCA`. Si la comunidad autónoma solicitada no se encuentra en el vector, devolverá el valor entero -1. **(1 punto)**

```
// Busca la componente del vector con codigoCA dada
??? buscaCA(vector<PoblacionVacunadosCA> ??v, int ??codigoCA){
    // Por completar
}
```

```
// Busca la componente del vector con codigoCA dada
int buscaCA(vector<PoblacionVacunadosCA> &v, int codigoCA){
    int valor = -1; // valor inicial, no encontrado
    bool encontrado = false;
    int i = 0;

    while(!encontrado && i < v.size()){
        if(v[i].codigoCA == codigoCA){
            encontrado = true;
            valor = i;
        }
        i++;
    }

    return valor;
}
```

2.c Ejercicio 3: Desarrolla un módulo (procedimiento o función) `leePoblacionYVacunados` que se encargue de leer el fichero de población y el fichero de vacunados, y devuelva un vector de tipos de dato `PoblacionVacunadosCA` con la información leída

. El módulo recibirá como parámetro dos cadenas, que serán los nombres de los ficheros que contienen la información a leer. Además, informará al usuario por pantalla con un mensaje de error (`"ERROR al abrir el fichero de población"` o `"ERROR al abrir el fichero de vacunados"`) en caso de que alguno de los ficheros no se abra correctamente. El mensaje de error permitirá conocer cuál ha sido el fichero que ha causado el error. **(1.75 puntos)**

> Tip
>
> puedes usar los módulos ya implementados que consideres necesarios.

```
// Lee el fichero de poblacion nombFPob y un fichero de vacunados nombFVac y
devuelve
// un vector de PoblacionVacunadosCA con la información dada por los ficheros
??? leePoblacionYVacunados(string ??nombFPob, string ??nombFVac){
    // Por completar
}
```

2.d Ejercicio 4: Desarrolla un módulo (procedimiento o función) `poblacionVacunadosMayor70` que se encargue de calcular cuáles son las comunidades autónomas con mayor y menor población total (no porcentual) vacunada en la cohorte 70-79+

. Este módulo recibirá un vector de `PoblacionVacunadosCA` y dos parámetros de tipo entero, `mayorCA` y `menorCA`, que contendrán después de la ejecución de este módulos los índices de las componentes del vector `PoblacionVacunadosCA` que tienen la mayor y la menor población total (no porcentual) vacunada en la cohorte 70-79+, respectivamente. **(1.25 puntos)**

```
// Calcula las comunidades autónomas con mayor y menor población total
// (no porcentual) vacunada en la cohorte 70-79+
??? poblacionVacunadosMayor70(vector<PoblacionVacunadosCA> ??vect,
                              int ??mayorCA, int ??menorCA){
    // Por completar
}
```

```
// Calcula las comunidades autónomas con mayor y menor población total
// (no porcentual) vacunada en la cohorte 70-79+
void poblacionVacunadosMayor70(vector<PoblacionVacunadosCA> &vect,
                               int &mayorCA, int &menorCA){
    // Paso por referencia del vector vect para evitar la copia

    mayorCA = 0;
    menorCA = 0;
    int ultimaCohorte = vect[0].vacunados.size() - 1;
```

```
    // Se puede tener definida una constante NUM_COHORTES con valor 7, y
    // acceder a NUM_COHORTES - 1

    for(int i = 1; i < vect.size(); i++){
        int aux = vect[i].vacunados[ultimaCohorte];
        if(aux > vect[mayorCA].vacunados[ultimaCohorte]){
            mayorCA = i;
        }
        if(aux < vect[menorCA].vacunados[ultimaCohorte]){
            menorCA = i;
        }
    }
}
```

2.e Ejercicio 5: Desarrolla un módulo (procedimiento o función) `resultadosPrevision` que se encargue de escribir en un fichero los datos de vacunados actualizados tras la previsión de vacunación

. Este módulo recibirá el nombre del fichero que se tiene que crear, un vector de `PrevisionVacunasCA` y un vector de `PoblacionVacunadosCA`, que representan la previsión de llegadas de viales y la población actual y vacunada hasta el momento de cada comunidad autónoma, respectivamente. El fichero tendrá el siguiente formato (observa que los campos están separados por coma):

```
COD_CA,0-17,18-29,30-39,40-49,50-59,60-69,70-79+
0,510,422,112,232,477,566,1990
2,110,212,44,432,747,866,990
1,180,242,144,932,547,966,1520
```

En caso de error durante la gestión del fichero, se avisará al usuario del mismo con una cadena informativa del error (mensaje `"ERROR al asociar el fichero de resultados"`). **(1.5 puntos)**

```
// Escribe en el fichero nombre los resultados de previsión
??? resultadosPrevision(string ??nombre, vector<PrevisionVacunasCA> ??vVac,
                                    vector<PoblacionVacunadosCA> ??vPobl){
    // Por completar
}
```

```
// Escribe en el fichero nombre los resultados de previsión
void resultadosPrevision(string nombre, vector<PrevisionVacunasCA> &vVac,
                                      vector<PoblacionVacunadosCA> &vPobl){
    // Paso por referencia de los vectores para evitar la copia
    ofstream f(nombre);
    int indCA;

    if(f.is_open()){
        // distribuimos los viales
      vector<PoblacionVacunadosCA> vPrevVac = distribuyeViales(vVac, vPobl);
```

```
        // escribimos cabecera
        f << "COD_CA,0-17,18-29,30-39,40-49,50-59,60-69,70-79+" << endl;

        // escribimos los datos de previsión de vacunación
        for(int i = 0; i < vPrevVac.size(); i++){
            f << vPrevVac[i].codigoCA << ',';
            // iteramos en el número de vacunados
            int j;
            for(j = 0; j < (vPrevVac[i].vacunados.size() - 1); j++){
                f << vPrevVac[i].vacunados[j] << ',';
            }
            // último elemento del vector
            f << vPrevVac[i].vacunados[j] << endl;
        }
    }else{
        cout << "ERROR al asociar el fichero de resultados" << endl;
    }
}
```

2.f Ejercicio 6: Desarrolla un módulo (procedimiento o función) `vacunados30_49` que se encargue de calcular el porcentaje de vacunados en España entre 30 y 49 años

. Este módulo recibirá un vector de `PoblacionVacunadosCA`, que representan la población actual y vacunada hasta el momento de cada comunidad autónoma, y devolverá el porcentaje de personas en España entre 30 y 49 años que se encuentran ya vacunadas. **(1.25 puntos)**

```
// Devuelve el porcentaje de personas en España entre 30 y 49 años ya vacunadas
??? vacunados30_49(vector<PoblacionVacunadosCA> ??vPobl){
    // Por completar
}
```

```
// Devuelve el porcentaje de personas en España entre 30 y 49 años ya vacunadas
float vacunados30_49(vector<PoblacionVacunadosCA> &vPobl){
    // Paso por referencia del vector para evitar la copia
    int totalPobl = 0;
    int vacunados = 0;
    for(int i = 0; i < vPobl.size(); i++){
        // 2 y 3 -> cohortes 30-39 y 40-49
        for(int j = 2; j < 4; j++){
            totalPobl += vPobl[i].poblacion[j];
            vacunados += vPobl[i].vacunados[j];
        }
    }

    return vacunados / (float)totalPobl;
}
```

2.g Ejercicio 7: Desarrolla un módulo (procedimiento o función) `previsionIncrementoViales` que, dados dos ficheros de previsiones de llegadas de viales, se encargue de escribir por pantalla el incremento de viales, en tanto por cierto, para cada marca de las vacunas aprobadas en cada comunidad autónoma.

Este módulo recibirá dos nombres de ficheros de previsiones, `primerMes` y `segundoMes`, y se encargará de calcular y escribir por pantalla el incremento de viales que se espera, en tanto por cierto y con dos decimales, de cada marca en cada comunidad autónoma. **(1.75 puntos)**

```
// Dados dos ficheros de previsiones, escribe por pantalla el incremento
// de viales en tanto por cierto para cada marca en cada comunidad autónoma
??? previsionIncrementoViales(string ??primerMes, string ??segundoMes){
    // Por completar
}
```

Un ejemplo de la salida esperada por pantalla es el siguiente (cada campo de información está separado por un carácter tabulador):

```
Com.Aut.    PZ  MD  JJ  AZ
ANDALUCÍA   41.20%  2.07%   6.00%   100.83%
NAVARRA -8.13%  32.90%  -4.47%  17.78%
ARAGÓN  -10.67% 10.37%  12.57%  -25.00%
```

```
// Dados dos ficheros de previsiones, escribe por pantalla el incremento
// de viales en tanto por cierto para cada marca en cada comunidad autónoma
void previsionIncrementoViales(string primerMes, string segundoMes){
    vector<PrevisionVacunasCA> vPrev1 = leePrevisiones(primerMes);
    vector<PrevisionVacunasCA> vPrev2 = leePrevisiones(segundoMes);

    cout << fixed << setprecision(2);
    cout << "Com.Aut.\tPZ\tMD\tJJ\tAZ" << endl;
    for(int i = 0; i < vPrev1.size(); i++){
        int codigoCA = vPrev1[i].codigoCA;
        cout << nombreDeCA(codigoCA) << '\t';

       // buscamos la componente en el otro vector (esto se puede hacer como
función)
        bool encontrado = false;
        int indCA, j = 0;
        while(!encontrado && j < vPrev2.size()){
            if(vPrev2[j].codigoCA == codigoCA){
                indCA = j;
                encontrado = true;
            }
            j++;
        }

        // mostramos datos solicitados
        cout << (vPrev2[indCA].numPF / (float) vPrev1[i].numPF  - 1)*100 <<
```

```
"%\t";
        cout << (vPrev2[indCA].numMD / (float) vPrev1[i].numMD  - 1)*100 <<
"%\t";
        cout << (vPrev2[indCA].numJJ / (float) vPrev1[i].numJJ  - 1)*100 <<
"%\t";
        cout << (vPrev2[indCA].numAZ / (float) vPrev1[i].numAZ  - 1)*100 <<
'%' << endl;
    }
}
```