Trabajo Práctico 2

El trabajo práctico número 2 tiene fecha de entrega para el día **12/07**, y está divido en tres partes:

Contenido

Introducción

En Islandia, la clínica Zyxcba Inc. organiza sus listas de espera a partir del año de inscripción como paciente. Así, dentro de cada especialidad se da prioridad a aquellos pacientes que se hayan inscripto hace más tiempo, pero también considerando la urgencia de la consulta. Para celebrar el 75 aniversario de la clínica, Zyxcba Inc. ha decidido remplazar la implementación actual del sistema de gestión en COBOL por una más moderna en C.

Funcionamiento de la clínica

Existe un listado de pacientes de la clínica en formato CSV; este listado incluye el año en el que fueron inscripto como pacientes.

Asimismo, existe otro listado CSV de todes les doctores, y la especialidad de cada une.

Les pacientes pueden pedir turno para una determinada especialidad, y el sistema les añade a la lista de espera correspondiente a esa especialidad. Les pacientes pueden venir de *urgencia*, o por un una revisión de rutina, guardia, etc. (se considera a todos los casos como *no urgentes*). Les pacientes *urgentes* se atienden por orden de llegada.

Cuando une doctore queda libre, atiende al siguiente paciente de urgencia, si es que los hubiera, o a le paciente con mayor antigüedad en la clínica.

Consigna

Se pide implementar el sistema de gestión de turnos y generación de informes de la clínica.

Cuando el sistema arranca, carga en memoria la lista de pacientes y la lista de les doctores. A partir de ese momento, lee de entrada estándar las operaciones a realizar (una por línea), tal y como se detalla a continuación.

Tras cada operación, el sistema siempre imprime por salida estándar un mensaje, informando de los resultados, o cualquier error que haya ocurrido.

Entradas al programa

El programa recibe como argumentos el nombre de **dos archivos en formato CSV**:

1. Archivo CSV con la lista de doctores. Tiene dos columnas: el nombre y la especialidad. Así:

```
nombre_doctor1,nombre_especialidad_A
nombre_doctor2,nombre_especialidad_B
nombre_doctor3,nombre_especialidad_A
...
```

Como se ve, cada doctore tiene solo una especialidad, pero una especialidad puede tener varies doctores.

2. Archivo CSV con la lista de pacientes. Tiene también dos columnas: el nombre y el año de su inscripción. Así:

```
nombre_paciente1,año_inscripcion
nombre_paciente2,año_inscripcion
```

Garantías (pre-condiciones):

- no hay nombres duplicados dentro de cada archivo (aunque une doctore y une paciente sí podrían llamarse igual).
- los nombres de doctor, nombre y especialidad no contienen ninguno de los siguientes caracteres: coma `','`, dos puntos `':'`, salto de línea `'\n'`, carácter nulo `'\0'`.
- los años de inscripción son números enteros positivos en base 10, y ningune paciente se inscribió luego del presente año, ni antes del año 0 (si bien la clínica cumple 75 años, tienen registros de clínicas anteriores que se fusionaron para crear la actual).

Datos disponibles

Se deja en del sitio de descargas un archivo con casos de prueba que pueden utilizar tanto de archivos de entrada, como de comandos a ser ejecutados y pruebas automatizadas, además de los restantes archivos que se mencionan de aquí en adelante.

Para ejecutar las pruebas correr:

```
$ ./pruebas.sh PATH-A-EJECUTABLE-TP2
```

Interacción

Una vez en marcha, el sistema queda a la espera de instrucciones, que le llegan por entrada estándar (`stdin`).

Una instrucción se compone de un comando y sus parámetros. El formato siempre es: nombre del comando, carácter dos puntos `':'` y parámetros del comando. Así:

```
    CMD1:ARG_A
    CMD2:ARG_B,ARG_C
```

Como se ve, un comando puede tener más de un parámetro, separados por comas.

Una vez procesada la operación, el sistema imprimirá el resultado, o un mensaje de error si no la pudo completar. **Tanto el resultado como los mensajes de error se imprimen siempre por salida estándar**. El programa puede por tanto utilizar la salida de error estándar para mensajes de depurado o *debugging*. Estos mensajes, no obstante, no deben quedar en el programa final.

Formato de los mensajes de error:

- Si una instrucción no sigue el formato `NOMBRE_COMANDO:PARAMETROS`, el sistema imprime el siguiente mensaje de error:

  ```
  ERROR: formato de comando incorrecto ('LINEA_INGRESADA')
  ```

- Si el formato es correcto pero no existe el comando, el sistema imprime:

  ```
  ERROR: no existe el comando 'NOMBRE_CMD'
  ```

- Si no se tienen la cantidad de parámetros necesarios, el sistema imprime:

  ```
  ERROR: cantidad de parametros invalidos para comando 'NOMBRE_CMD'
  ```

Comandos a implementar

El sistema implementa los tres comandos que se detallan a continuación.

1. Pedir turno

Se recibe un nombre de paciente y el nombre de una especialidad, y el sistema le añade a la lista de espera de la especialidad correspondiente.

Formato:

```
PEDIR_TURNO:NOMBRE_PACIENTE,NOMBRE_ESPECIALIDAD,URGENCIA
```

Los valores válidos para `URGENCIA` son: `URGENTE` o `REGULAR`.

Salida (dos líneas):

```
Paciente NOMBRE_PACIENTE encolado
N paciente(s) en espera para NOMBRE_ESPECIALIDAD
```

Mensajes de error, cuando corresponda (imprimir cada mensaje por cada error encontrado, no detenerse ante el primer error):

```
ERROR: no existe le paciente 'NOMBRE_PACIENTE'
ERROR: no existe la especialidad 'NOMBRE_ESPECIALIDAD'
ERROR: grado de urgencia no identificado ('URGENCIA')
```

Garantías (pre-condiciones):

- une paciente nunca solicitará turno para una especialidad si ya está en la lista de espera de la misma. No obstante, puede estar simultáneamente en la lista de espera de dos o más especialidades distintas.

2. Atender siguiente paciente

Se recibe el nombre de le doctore que quedó libre, y este atiende al siguiente paciente urgente (por orden de llegada). Si no hubiera ningún paciente urgente, atiende al siguiente paciente con mayor antigüedad como paciente en la clínica.

Formato:

```
ATENDER_SIGUIENTE:NOMBRE_DOCTOR
```

Salida si se atendió a une paciente (dos líneas):

```
Se atiende a NOMBRE_PACIENTE
N paciente(s) en espera para NOMBRE_ESPECIALIDAD
```

donde N es el número de pacientes que quedaron en espera tras atender al paciente actual (que puede ser 0).

Salida si no había previamente pacientes en la lista de espera (una línea):

```
No hay pacientes en espera
```

Mensajes de error, cuando corresponda:

```
ERROR: no existe el doctor 'NOMBRE_DOCTOR'
```

3. Informe doctores

El sistema imprime la lista de doctores **en orden alfabético**, junto con su especialidad y el número de pacientes que atendieron desde que arrancó el sistema. Opcionalmente, se puede especificar el rango (alfabético) de doctores sobre los que se desean informes.

Formato:

```
INFORME:[INICIO],[FIN]
```

En caso de no quererse indicar un inicio o fin, simplemente se deja vacío, por ejemplo:

```
INFORME:Gonzalez,Gutierrez
INFORME:,Gutierrez
INFORME:Gonzalez,
INFORME:,
```

Salida ($N + 1$ líneas, donde N es el número de doctores en el sistema que se encuentran en dicho rango):

```
N doctor(es) en el sistema
1: NOMBRE, especialidad ESPECIALIDAD, X paciente(s) atendido(s)
2: NOMBRE, especialidad ESPECIALIDAD, Y paciente(s) atendido(s)
...
N: NOMBRE, especialidad ESPECIALIDAD, Z paciente(s) atendido(s)
```

Mensajes de error

Para facilitar los mensajes de la aplicación sean *exactamente* los especificados, se proporciona un archivo `mensajes.h` que contiene las definiciones de todos los mensajes, tal que:

```
#define PACIENTE_ENCOLADO "Paciente %s encolado\n"
#define ENOENT_DOCTOR "ERROR: no existe el doctor '%s'\n"
// etc.
```

La manera de usarlos, entonces, es:

```
#include "mensajes.h"
#include <stdio.h>

printf(PACIENTE_ENCOLADO, "Ingeniero Barrios");
```

Si hay alguna discrepancia entre esta consigna y el archivo `mensajes.h`, este último es la versión canónica.

Otras condiciones de error

El sistema aborta con código numérico 1 ante cualquiera de los siguientes errores **durante la fase de inicialización**:

- no se recibieron exactamente dos argumentos por la línea de comandos;
- alguno de los archivos CSV no existe, o no pudo ser leído;
- algún año no es un valor numérico.

Funciones auxiliares del curso

Además del archivo `mensajes.h` ya mencionado, se proporcionan dos archivos auxiliares, `csv.h` y `csv.c`, para ayudar con la lectura tanto de los archivos CSV, como de los comandos. Su uso es opcional; cada grupo puede emplear sus propias funciones de lectura, o las proporcionadas por el curso en TPs anteriores.

Además, se añade un esqueleto del programa principal del Trabajo Práctico `zyxcba.c`. Pueden bien utilizarlo, en cuyo caso deberán modificarlo para adaptar su uso a la estructuras de datos que utilicen. No es obligatorio su uso.

Criterios de aprobación

Los siguientes aspectos son condición necesaria para la aprobación del trabajo práctico.

Estructuras de datos

Es necesario emplear la estructura de datos más apropiada para cada función del programa, en particular **teniendo en cuenta la complejidad temporal**. También es necesario utilizar dicha estructura de la forma más apropiada posible para optimizar de manera correcta el funcionamiento de la clínica.

Se puede (y alentamos) implementar otros TDAs o estructuras que faciliten o mejoren la implementación de este Trabajo Práctico. Les recordamos que un TDA no es un simple `struct` donde se guarda información, sino que debe tener comportamiento. Esto será un punto muy importante en la evaluación del Trabajo Práctico.

Todas las estructuras deben estar implementadas de la forma más genérica posible y correctamente documentadas. En general, pueden emplear sin modificar los tipos implementados en entregas anteriores en la materia. Potencialmente, deberán modificar el TDA Árbol Binario de Búsqueda. En cualquier caso, si fuera necesario modificar algún TDA, la funcionalidad agregada debe estar implementada de forma completamente abstracta al presente trabajo práctico.

Código del programa

El código entregado debe:

- ser claro y legible, y estar estructurado en funciones lo más genéricas posible, adecuadamente documentadas.
- compilar sin advertencias y correr sin errores de memoria.

- ajustarse a la especificación de la consigna y pasar todas las pruebas automáticas.

Complejidad

Cada comando debería funcionar de forma acorde, pero además deberá cumplir con los siguientes requisitos de complejidad:

1. Comando Pedir Turno: Si se tratase de un caso urgente, deberá funcionar en $\mathcal{O}(1)$. Si, en cambio, se tratara de un caso regular, se puede relajar a que funcione en $\mathcal{O}(\log n)$ siendo n la cantidad de pacientes encolados **en la especialidad** indicada (no el total de los pacientes existentes).

2. Comando Atender siguiente paciente: Siendo d la cantidad de doctores en el sistema, debe funcionar en $\mathcal{O}(\log d)$ si el siguiente caso a tratar en la especialidad de dicho doctor es un caso urgente, o bien $\mathcal{O}(\log d + \log n)$ si se tratara de un caso regular.

3. Comando Informe doctores: debe ser $\mathcal{O}(d)$ en el peor caso (en el que se tenga que mostrar todos los doctores del sistema), $\mathcal{O}(\log d)$ en un caso promedio (en el caso en el que no se pidan mostrar demasiados doctores).

Informe

Este trabajo práctico tiene como principal foco en el diseño del mismo (modularización, creación de nuevas estructuras y TDAs específicos del TP) así como el uso apropiado de las estructuras de datos vistas en clase. Por lo tanto, se les pide que escriban un informe de dicho diseño, puesto que además de ser capaces de escribir código correcto, también es necesario que aprendan a poder *transmitir* el porqué de sus decisiones, utilizando el lenguage técnico y justificaciones correspondientes.

El informe deberá consistir de las siguientes partes:

- carátula con los datos personales del grupo, y ayudante asignado.

- análisis y diseño de la solución, en particular: algoritmos y estructuras de datos utilizados, justificando conforme a los requisitos pedidos en este TP.

Entrega

La entrega incluye, obligatoriamente, los siguientes archivos:

- El código de los TDAs utilizados.
- El código del TP.
- El informe (en formato `.pdf`).
- Un archivo *deps.mk* que exprese las dependencias del proyecto en formato makefile. Este archivo deberá contener solamente dos líneas que indiquen, para cada programa, de qué *objetos* depende su ejecutable; por ejemplo:

  ```
  # Ejemplo de archivo deps.mk para el TP2
  zyxcba: zyxcba.o csv.o hash.o
  ```

La entrega se realiza exclusivamnete en forma digital a través del sistema de entregas, con todos los archivos mencionados en un único archivo ZIP.