INF-134 Estructuras de Datos, 2022-1

Tarea 1

Profesores: Juan Pablo Castillo, Roberto Díaz, Javier Robledo
juan.castillog@sansano.usm.cl, roberto.diazu@usm.cl, javier.robledo@usm.cl

Ayudantes:
Domingo Benoit (domingo.benoit@sansano.usm.cl)
Joaquín Gatica (joaquin.gatica@sansano.usm.cl)
Diana Gil (diana.gil@sansano.usm.cl)
Martín Ruiz (martin.ruiz@usm.cl)
Martín Salinas (martin.salinass@sansano.usm.cl)
Betrice Valdes (beatrice.valdes@sansano.usm.cl)

Fecha de entrega: 16 de abril, 2022.
Plazo máximo de entrega: 5 días.

1. Reglas del Juego

La presente tarea debe hacerse en grupos de 3 personas. Toda excepción a esta regla debe ser conversada con los ayudantes (usando los correos indicados en el encabezado de esta tarea). No se permiten de ninguna manera grupos de más de 3 personas. Debe usarse el lenguaje de programación C++. Al evaluarlas, las tareas serán compiladas usando el compilador `g++`, usando la línea de comando `g++ archivo.cpp -o output -Wall`. Alternativamente, se aceptan variantes o implementaciones particulares de `g++`, como el usado por MinGW (que está asociado a la IDE code::blocks). Se deben seguir los tutoriales que están en Aula USM, cualquier alternativa explicada allí es válida. Recordar que una única tarea en el semestre puede tener nota menor a 30. El no cumplimiento de esta regla implica reprobar el curso. No se permite usar la biblioteca *stl*, así como ninguno de los *containers* y algoritmos definidos en ella (e.g. `vector`, `list`, etc.). Está permitido usar otras funciones de utilidad definidas en bibliotecas estándar, como por ejemplo `math.h`, `string`, `fstream`, `iostream`, etc.

2. Objetivos

Comprender y familiarizarse con las estructuras y tipos de datos básicos que provee el Lenguaje de Programación C++. Entre los conceptos mas importantes, se encuentran:

- Paso de parámetros por valor.
- Paso de parámetros por referencia.
- Asignación de memoria dinámica.
- Manipulación de punteros.
- Manejo de Archivos.

Además se fomentará el uso de las buenas prácticas y el orden en la programación de los problemas correspondientes.

3. Problemas a Resolver

En esta sección se describen los problemas a resolver implementando programas, funciones o clases en C++. Se debe entregar el código para cada uno de los problemas en archivos `.cpp` separados (y correspondientes `.hpp` de ser necesario).

3.1. Reporte de Comedor

En una cierta empresa, un comedor ocupa dispositivos con reconocimiento facial para la validación de funcionarios y la correcta entrega de tickets de almuerzo. El funcionario se acerca a alguno de los dispositivos y este imprime el ticket si es que el servidor valida que no haya sacado previamente alguno. Lamentablemente por un error en la red, los dispositivos se encontraban funcionando en un modo offline y es posible que algunos tickets hayan sido otorgados incorrectamente. Su tarea es detectar cuántos tickets de comedor fueron repartidos incorrectamente para calcular los costos.

3.1.1. Entrada

El comedor provee uno o más **servicios** que constan de:

- Nombre del servicio
- Límite de tickets diario por funcionario por servicio
- Límite de tickets mensual por funcionario por servicio
- Hora de inicio
- Hora de fin

Cada funcionario podrá ser identificado por su RUT y tendrá un **límite de 100 tickets mensuales totales** que será independiente de los servicios y límites señalados anteriormente.

Los datos de los servicios se han rescatado desde el servidor y se encuentran en un archivo de texto en formato ASCII llamado `servicios.txt`. Este archivo tiene la siguiente estructura: la primera línea contiene un número entero n con el número de servicios provistos, seguido de n líneas en que cada una contiene la información de un único servicio: un string con el nombre, dos enteros con los límites de ticket diario y mensual respectivamente, y dos strings en formato `hh:mm` representando las horas de inicio y fin de cada servicio en formato de 24 horas. Cada componente del servicio estará separada por un espacio de la siguiente.

Un ejemplo de este archivo se presenta a continuación:

```
Desayuno 1 30 06:00 10:00
Almuerzo 1 30 11:30 15:00
Once 2 60 18:00 19:00
Cena 1 30 21:00 22:00
Nocturno 1 30 23:30 03:00
```

Por otro lado, los datos de un ticket obtenido por algún funcionario se encuentran descritos por el `struct` siguiente:

```
struct Ticket {
    char rut_funcionario[10];
    int day_of_month;
    char time[6];
};
```

Donde el campo `rut_funcionario` es un string que contiene el RUT del funcionario, `day_of_month` un entero entre 1 y 31 que representa el día del mes y `time` un string en formato `hh:mm` que representa la hora y minuto de emisión del ticket.

Los datos de los tickets emitidos el último mes se encuentran en un archivo binario llamado `tickets.dat`. La estructura de este archivo es la siguiente: un entero m indicando el número de tickets, seguido de m structs de tipo `Ticket`.

3.1.2. Salida

Su programa, según la información leída desde los archivos anteriores, debe detectar aquellos tickets emitidos inválidamente y contabilizarlos para cada funcionario.

Tome en consideración los siguientes casos a detectar:

- Cualquier ticket emitido fuera del horario de cualquier servicio
- Un funcionario excede el límite de tickets diarios de un servicio
- Un funcionario excede el límite de tickets mensuales de un servicio
- Un funcionario excede el límite de tickets mensuales totales

La salida de su programa deberá ser por salida estándar, mostrando en líneas separadas para cada funcionario: su RUT, seguido del número de tickets emitidos válidamente para el funcionario, un slash ('/') y el número de tickets totales emitidos para el funcionario. La salida deberá estar ordenada lexicográficamente por el RUT. Un ejemplo de salida es el siguiente:

```
041069112 30/30
06753719K 0/30
180845887 15/20
248950129 30/120
```

3.1.3. Consideraciones

- El nombre de cada servicio no contendrá espacios ni otros caracteres en blanco
- Los consumos realizados fuera del horario de cualquier servicio no contaran para los límites de ticket máximo diarios o mensuales de ningún servicio
- Los consumos en el minuto de inicio y término de un servicio se consideran válidos (i.e. un ticket emitido a las 12:00 es válido para un servicio que comienza (o termina) a las 12:00)
- Un servicio puede comenzar antes y terminar después de la medianoche. Un funcionario no podrá exceder el límites de tickets diarios durante un mismo periodo de servicio contiguo, pero si podrá sacar tickets cuando el servicio se reinicie. Para el caso del servicio nocturno presentado en el ejemplo anterior (que empieza a las 23:30 y termina a las 03:00, con máximo 1 ticket diario por servicio), un funcionario no podrá sacar un ticket a las 23:45 del día 1 y a las 02:00 del día 2; pero sí podrá sacar un ticket a las 02:00 del día 1 y a las 23:45 del día 1.
- Los límites de tickets diarios y mensuales pueden variar por servicio.
- Las horas y minutos siempre estarán en formato de 2 dígitos.
- El RUT de cada funcionario incluirá el dígito verificador, pero omitirá los puntos y el guión. Los RUTs mas cortos se rellenarán con ceros a la izquierda. La `K` para el dígito verificador siempre será mayúscula.

4. Entrega de la Tarea

Entregue la tarea enviando un archivo comprimido `tarea1-apellido1-apellido2-apellido3.zip` o `tarea1-apellido1-apellido2-apellido3.tar.gz` (reemplazando sus apellidos según corresponda) a la página aula.usm del curso, a más tardar el día 16 de abril, 2022, a las 23:59:00 hs (Chile Continental), el cual contenga como mínimo:

- Los archivos con los códigos fuentes necesarios para el funcionamiento de la tarea. ¡Los archivos deben compilar!
- **nombres.txt**, Nombre, ROL, Paralelo y qué programó cada integrante del grupo.
- **README.txt**, Instrucciones de compilación en caso de ser necesarias, y la forma de compilación que usó (debe ser alguna de las indicadas en los tutoriales entregados en Aula USM).

5. Restricciones y Consideraciones

- Por cada día de atraso en la entrega de la tarea se descontarán 10 puntos en la nota.
- El plazo máximo de entrega es 5 días después de la fecha original de entrega.
- **Las tareas que no compilen no serán revisadas y serán calificadas con nota 0**.
- Debe usar **obligatoriamente** alguna de las formas de compilación indicadas en los tutoriales entregados en Aula USM.
- Por cada *Warning* en la compilación se descontarán 5 puntos.
- Si se detecta **COPIA** la nota automáticamente sera 0 (CERO), para todos los grupos involucrados. El incidente será reportado al jefe de carrera.
- La prolijidad, orden y legibilidad del código fuente es obligatoria. Habrá descuentos si alguno de estos items no se cumple.

6. Consejos de Programación

El código fuente del programa debe estar estructurado adecuadamente en archivos (separados de ser necesario). Si el código fuente está desordenado, se pueden descontar hasta 20 puntos de la nota.

Cada función programada debe tener comentarios de la siguiente forma:

```
/*****
*   TipoFunción NombreFunción
******
*   Resumen Función
******
*   Input:
*       tipoParámetro NombreParámetro : Descripción Parámetro
*        .......
******
*   Returns:
*       TipoRetorno, Descripción retorno
*****/
```

Por cada comentario faltante, se restarán 5 puntos.

Por último, la indentación (1 TAB o 4 espacios), es muy importante. Por **cada bloque mal indentado, se quitarán 10 puntos**.

7. Bonus

Los grupos que realicen su propia implementación **eficiente** del algoritmo de ordenamiento estarán participando por un libro de Estructuras de Datos y obtendrán hasta 10 puntos extras en la nota de esta tarea.