INF-134 Estructuras de Datos, 2023-1

Tarea 1

Profesores: Gabriel Carmona, Juan Pablo Castillo, Roberto Díaz, Hubert Hoffmann
gabriel.carmonat@sansano.usm.cl,
juan.castillog@sansano.usm.cl,
roberto.diazu@usm.cl
hoffmann@inf.utfsm.cl

Ayudantes San Joaquín:
Diego Debarca
diego.debarca@sansano.usm.cl
Gabriel Escalona
gabriel.escalona@usm.cl
Joaquín Gatica
joaquin.gatica@sansano.usm.cl
Juan Cucurella
juan.Cucurella@usm.cl
Rodrigo Flores
rodrigo.floresf@sansano.usm.cl

Ayudantes Casa Central:
Catalina Cortez
catalina.cortez@usm.cl
Benjamin Cayo
benjamin.cayo@usm.cl
Sebastián Torrealba
sebastian.torrealba@usm.cl
Tomás Barros
tomas.barros@sansano.usm.cl

Fecha de entrega: Lunes 17 de Abril.
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`. **No se aceptarán variantes o implementaciones particulares de g++, como el usado por MinGW**. 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 deben implementarse funciones y clases en C++, de acuerdo a las siguientes descripciones. Se debe entregar cada uno de los problemas en archivos `.cpp` separados (y correspondientes `.hpp` de ser necesario). Para cada problema, entregue una función `main` adecuada que permita probar sus programas. Sin embargo, tenga en cuenta que durante la corrección se probarán otras funciones `main`.

3.1. Casino de Alimentación

Una cierta empresa ha adquirido unos dispositivos con reconocimiento facial para la entrega de tickets de alimentación en el casino. Un colaborador podrá acercarse a uno de estos dispositivos y obtener su ticket si no ha superado su saldo diario por servicio. La obtención de un ticket se considerará como un "consumo".

Cada colaborador podrá sacar un número de tickets limitado por su saldo en cada servicio del casino (desayuno, almuerzo, once y cena). Dicho saldo puede variar para cada colaborador y servicio.

El dispositivo al captar un rostro, tratará de emparejarlo con un colaborador. En caso de reconocerlo, entregará su RUT, en caso contrario, el *string* vacío. El RUT debe ser usado para la verificación del saldo y registrar el consumo.

3.1.1. Consideraciones

Considere las siguientes constantes que permiten identificar cada uno de los servicios de alimentación.

```
int const SERV_DESAYUNO = 0;
int const SERV_ALMUERZO = 1;
int const SERV_ONCE = 2;
int const SERV_CENA = 3;
```

Considere el `struct SaldoColaborador` presentado a continuación que almacena para un colaborador el saldo para cada uno de los servicios. Los valores de saldo en este struct no se actualizan, sino que solo se consultan. El valor de cada saldo será mayor o igual a cero.

```
struct SaldoColaborador {
    string rut;
    int saldo_desayuno;
    int saldo_almuerzo;
    int saldo_once;
    int saldo_cena;
}
```

El dispositivo registra los consumos de cada día en un archivo ASCII con un nombre arbitrario. En cada línea del archivo se guardan, separados por espacios, el RUT y el servicio en que se realizó ese consumo. Para cada servicio se guarda el nombre del servicio en mayúsculas. Notar que este archivo es creado al registrar el primer consumo del día. A continuación se entrega un ejemplo de este archivo con cinco consumos:

```
13314801-9 DESAYUNO
13314801-9 ALMUERZO
03686224-6 ONCE
78927357-k CENA
03686224-6 ONCE
```

Por otro lado, el dispositivo guarda la información de los saldos de cada colaborador en un archivo binario llamado `saldos.bin` que primero guarda un entero n y luego n structs `SaldoColaborador` ordenados lexicográficamente. En caso de que un colaborador no exista en este archivo se debe considerar como que tiene saldo cero en todos los servicios.

3.1.2. Requerimiento

Se debe implementar la función `bool puedeConsumir(string rut, int servicio, string consumos_dia` que retorna verdadero si es que el colaborador tiene saldo para recibir un ticket de alimentación y falso en otro caso. Un colaborador no tendrá saldo si es que el número de tickets anteriores en el servicio es igual o mayor a su saldo. En caso de tener saldo también debe registrar el consumo en el archivo de consumos del día. Esta función recibe los siguientes parámetros:

- `rut`: Un string con el RUT del colaborador en formato `NNNNNNNN-D` con cada `N` siendo un número del `0` al `9` y `D` un número del `0` al `9` o la letra `k` (e.g. `13314801-9`, `03686224-6`, `78927357-k`).
- `servicio`: un entero que identifica el servicio del consumo y que debe ser una de las constantes `SERV_*` presentadas en la sección anterior.
- `consumos_dia`: un string con el nombre de archivo que contiene los consumos del día.

3.2. Listas de la suerte

En el mundo Cplusplusiano, cada persona puede comprar una vez por día una tarjeta de la suerte la cual es generada a base del nombre de la persona. Además, una persona puede intercambiar las tarjetas de la suerte con otra persona. Luego, al final del día se utiliza la fecha de nacimiento y la tarjeta para calcular el puntaje de la suerte de una persona.

3.2.1. Consideraciones

- El `struct Persona` presentado a continuación que almacena:
  - `string nombre`: corresponde al nombre de la persona
  - `char fecha[11]`: corresponde a la fecha de nacimiento de la persona con el siguiente format `AAAA-MM-DD`, por ejemplo `1999-01-10`
  - `int tamanio_tarjeta`: corresponde a la cantidad de numeros en la tarjeta
  - `int* tarjeta`: corresponde a un punteo a un arreglo de enteros
  - `bool quiere_intercambiar`: corresponde a un booleano indicando si quiere intercambiar o no la persona

```
struct Persona {
    string nombre;
    char fecha[11];
    int tamanio_tarjeta;
    int* tarjeta;
    bool quiere_intercambiar;
}
```

Las personas serán ingresadas por entrada estándar con el siguiente formato:

```
p
nombre_1 fecha_1 quiere_intercambiar_1
nombre_2 fecha_2 quiere_intercambiar_2
.
.
.
nombre_p fecha_p quiere_intercambiar_p
```

Las personas inicialmente no tienen tarjeta de la suerte. El `nombre` no puede contener espacios o caracteres especiales. `quiere_intercambiar` será 0 o 1 (donde 0 es que no quiere y 1 es que si quiere).

- Para que la persona en la posición i intercambie su tarjeta con otra persona se debe cumplir que:
  - La persona en la posición i debe querer intercambiar.
  - Debe buscar una persona en otra posición que quiera intercambiar, además esa persona debe tener algún número en común con la persona en la posición i.

  Si el intercambio se realiza, la persona en la posición i deja de querer intercambiar.
- En caso de empate, la persona en la menor posición se considera el ganador.

3.2.2. Requerimientos

Se debe implementar las siguientes funciones:

- `int* comprarTarjeta(string nombre, int dia, int &m)`: recibe dos parámetros el nombre de la persona que quiere comprar, el día en que se compra la tarjeta y el tamaño del arreglo que se va a retornar. Debe retornar un arreglo de enteros de igual tamaño que el nombre, dónde en la posición i se encontrará el valor $nombre[i]\,\%dia$.
- `void intercambiarTarjeta(Persona* p1, Persona* p2)`: recibe dos personas como parámetro se debe intercambiar las tarjetas entre las dos personas.
- `int puntaje(Persona* p1)`: recibe una persona y debe retornar un entero correspondiente al puntaje de la persona, que corresponderá al siguiente valor:

$$\sum_{i=0}^{tamanio\_tarjeta-1} p1 \to tarjeta[i] * p1 \to fecha[i\,\%10]$$

- `Persona* unDia(Persona* personas, int dia)`: recibe un arreglo de personas, debe comprar una tarjeta a cada persona utilizando el parámetro `dia`, luego que cada persona intente intercambiar hasta que sea imposible seguir intercambiando tarjetas. Se debe partir siempre por la persona en la posición 0 hasta la persona en la posición $p-1$. Finalmente, debe retornar la persona que haya obtenido el mayor puntaje.
- `void variosDias(Persona* personas, int cant_dias)`: recibe un arreglo de personas y la cantidad de dias en total a realizar. Debe llamar a la función `unDia` pasando por parámetro los días 1 hasta `cant_dias`. Por cada día, debe mostrar por pantalla el nombre, fecha de nacimiento y puntaje de la persona que obtuvo el mayor ese día.

3.2.3. Ejemplo

Se entrega por entrada estándar el siguiente caso:

```
4
Pepito 1999-01-12 1
Juanito 2000-12-10 0
sinnombre 2012-20-01 1
puntoycoma 2012-20-20 0
```

Se llama a la función `variosDias` con esa listas de personas y una cantidad de días equivalente a 2.
El primer día las tarjetas de las personas son:

- Pepito: `[0, 0, 0, 0, 0, 0]`

- Juanito: `[0, 0, 0, 0, 0, 0, 0]`
- sinnombre: `[0, 0, 0, 0, 0, 0, 0, 0, 0]`
- puntoycoma: `[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]`

Pepito quiere intercambiar y tiene al menos un número en común con sinnombre, entonces intercambia con él. Dejando las tarjetas de las personas de la siguiente forma:

- Pepito: `[0, 0, 0, 0, 0, 0, 0, 0, 0]`
- Juanito: `[0, 0, 0, 0, 0, 0, 0]`
- sinnombre: `[0, 0, 0, 0, 0, 0]`
- puntoycoma: `[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]`

sinnombre también quiere intercambiar, pero nadie más quiere intercambiar de las personas, por lo que las tarjetas quedan igual.

Calculando el puntaje de todas las personas, todos valen 0. Por lo tanto el ganador de ese día es Pepito. Por lo que se debe mostrar por pantalla:

```
Pepito 1999-01-12 0
```

El segundo día las tarjetas de las personas son:

- Pepito: `[0, 1, 0, 1, 0, 1]`
- Juanito: `[0, 1, 1, 0, 1, 0, 1]`
- sinnombre: `[1, 1, 0, 0, 1, 1, 0, 0, 1]`
- puntoycoma: `[0, 1, 0, 0, 1, 1, 1, 1, 1, 1]`

Pepito quiere intercambiar y tiene al menos un número en común con sinnombre, entonces intercambia con él. Dejando las tarjetas de las personas de la siguiente forma:

- Pepito: `[1, 1, 0, 0, 1, 1, 0, 0, 1]`
- Juanito: `[0, 1, 1, 0, 1, 0, 1]`
- sinnombre: `[0, 1, 0, 1, 0, 1]`
- puntoycoma: `[0, 1, 0, 0, 1, 1, 1, 1, 1, 1]`

sinnombre también quiere intercambiar, pero nadie más quiere intercambiar de las personas, por lo que las tarjetas quedan igual.

Se calcula el puntaje de cada persona y se obtiene:

- Pepito: 241
- Juanito: 191
- sinnombre: 148
- puntoycoma: 334

Por lo tanto el ganador de ese día es puntoycoma. Entonces, se debe mostrar lo siguiente por pantalla:

```
puntoycoma 2012-20-20 334
```

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 Lunes 17 de Abril, a las 23:59:00 hs (Chile Continental), el cual contenga:

- 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**.